Generalisation & Optimisation in Neural Networks

Christopher Mingard

The Queens' College

University of Oxford

This dissertation is submitted for the degree of

Doctor of Philosophy in Theoretical Chemistry

October 2025

Acknowledgements

I am enormously grateful to Ard, for providing me with the opportunity to do a project on Machine Learning in my second year, and continuing (despite, I am sure, his better judgment) to supervise me on further research throughout the rest of my undergrad degree, and on into the DPhil. Thanks for the long discussions, guidance, knowledge, careful critiques of my ideas, and for accommodating my absence during the university cricket season. Thanks to Pat Roche, Sid Parameswaran, and Sam Henry for teaching me physics during my undergraduate studies and showing me the levels of rigorous thinking required to succeed in research.

I am grateful to the members of the Louis group. Yoonsoo, for wide-ranging discussions on feature learning (and many other topics) as well as his excellent cooking. Nic, for his sense of humour and enthusiasm for statistical physics. Lukas, for great ideas and the ability to find errors everywhere. Thanks also to Guillermo for introducing me to machine learning and for being an endless source of insight on all things Gaussian processes, and to Charlie for discussing nearly every project I have undertaken in the past few years. Thanks also to Sofia, Henry, Nayara, and Jessica. Thanks to my other co-authors, including Johnny, Soufiane, and Jeremy, for introducing me to the world of optimisation. I am also grateful to Vlad and Joar for helping steer me from physics toward computer science; to Vlad (2) and Richie for showing me how to write code properly; and to Sean for the many hours we spent debating proofs and priors.

Playing cricket for Oxford was the most important part of my time at university. I will always be grateful to everyone at OUCC who made the club possible, and for sharing in some excellent memories. It is not possible to list everyone, but to name a few, Ella, Owen, Dan, Robbo, Saq, Clarkey, Toby, Rosie and Coach for spending many hours analysing games, picking XIs, beating Cambridge and much else besides. Huge thanks to my family for their unwavering support and for watching varsity matches, despite the stressful close finishes, and bad weather.

Having spent the best part of a decade at Oxford, there are far too many people to thank for making the time enjoyable. To name a few, though, Anna, Pippa, Anita, Ferdy, Steven, Ruggero, Blanca and Naomi.

This thesis was made possible by an iCASE grant with IBM.

Abstract

The goal of this thesis is to contribute to our understanding of generalisation and optimisation in neural networks. We ask the following questions:

1. Why can highly expressive neural networks learn functions that generalise?
2. Optimiser hyperparameters can significantly affect generalisation. Why? How can we derive optimisers in a principled manner to maximise performance?

In the first part, we introduce a discrete fully-connected network (DFCN) model that offers useful insights for both questions. We prove a one-to-one correspondence between DFCN architectures and Disjunctive Normal Form (DNF) boolean expressions. This yields an interpretable complexity measure, $K_{DNF}(f)$ (shortest DNF length), which maps to the network's minimum weight norm. We show the prior over functions, $P(f)$, exponentially favors functions with low $K_{DNF}(f)$. Consequently, low-complexity functions are learnable, while high-$K_{DNF}(f)$ functions are not. Finally, we show that weight decay enhances this simplicity bias, acting as a penalty on $K_{DNF}(f)$ to promote learning minimal DNF representations and significantly improve generalisation.

In the second part, we focus on Question 1, through studying inductive biases towards simple functions inherent at initialisation. While explicit in Bayesian posteriors, we demonstrate our predictions empirically extend to networks trained with standard optimisers. We analyse the prior of neural networks on Boolean data. We find the prior, $P(f) \lesssim 2^{-aK(f)}$, can be controllably weakened by tuning the initial weight variance, σ_w, to move the network from an ordered ($a = 1$) to a chaotic ($a < 1$) regime. The latter leads to poor generalisation as the prior cannot counteract the functional growth. Then we reveal this architectural bias follows a universal pattern: the prior probability of a function adheres to Zipf's Law ($P(f) \propto R(f)^{-1}$). We then prove this Zipfian prior is a necessary condition for efficient learning.

Finally, in the third part, we focus on Question 2. We show that for wide networks, stochastic optimisers approximate Bayesian inference, and explore when this breaks down, leading to feature learning. We then introduce a framework to quantify feature learning, analyzing how optimizers and architectures impact learned representations. Then, we derive a new optimizer from first principles. We extend mirror descent to incorporate neural architecture, yielding Automatic Gradient Descent (AGD): a first-order, hyperparameter-free optimizer that trains networks at ImageNet scale and provides a foundation for new architecture-aware algorithms.

Originality, Publications, and Presentations

I have written the following papers, in order of date, that are related to either the question of optimisation or generalisation in neural networks. For the papers submitted as part of the thesis, contributions have been highlighted.

† Mingard, C., Skalse, J., Valle-Pérez, G., Martínez-Rubio, D., Mikulik, V. and Louis, A.A., 2019. Neural networks are a priori biased towards boolean functions with low entropy. arXiv preprint arXiv:1909.11522. [1]

† Mingard, C., Valle-Pérez, G., Skalse, J. and Louis, A.A., 2021. Is SGD a Bayesian sampler? Well, almost. Journal of Machine Learning Research, 22(79), pp.1-64. [2]

† Lou, Y., Mingard, C. and Hayou, S., 2022, June. Feature learning and signal propagation in deep neural networks. In International Conference on Machine Learning (pp. 14248-14282). PMLR. [3]

Bernstein, J.*, Mingard, C.*, Huang, K., Azizan, N. and Yue, Y., 2023. Automatic gradient descent: Deep learning without hyperparameters. arXiv preprint arXiv:2304.05187. [4]. *Conception from J.B. Derivations largely done by J.B. with help from C.M. Experiments done by C.M.*

Mingard, C., Rees, H., Valle-Pérez, G. and Louis, A.A., 2023. Deep neural networks have an inbuilt Occam's razor. Nature Communications, 16(1), p.220. [5]. *Project conception by C.M. H.R. and A.A.L. Experimental work and theory joint work by C.M. and H.R.*

† Nam, Y., Fonseca, N., Lee, S.H., Mingard, C. and Louis, A., 2024. An exactly solvable model for emergence and scaling laws in the multitask sparse parity problem. Advances in Neural Information Processing Systems, 37, pp.39632-39693. [6]

† Mingard, C.*, Pointing, J.*, London, C., Nam, Y. and Louis, A.A., 2024. Exploiting the equivalence between quantum neural networks and perceptrons. arXiv preprint arXiv:2407.04371. [7]

Nam, Y., Mingard, C., Lee, S.H., Hayou, S. and Louis, A., 2024. Visualising Feature Learning in Deep Neural Networks by Diagonalizing the Forward Feature Map. arXiv preprint arXiv:2410.04264. [8]. *Project conception by Y.N., theory and discussion joint work by Y.N. and C.M., vast majority of experiments by Y.N.*

Mingard, C.*, Seier, L.*, Göring, N., Badelita, A.V., London, C. and Louis, A., 2025. Characterising the Inductive Biases of Neural Networks on Boolean Data. arXiv preprint arXiv:2505.24060. [9]. *Project conception by C.M., majority of theory and experiments done by C.M. with contributions from all other co-authors.*

† Göring, N.A., London, C., Erturk, A.H., Mingard, C., Nam, Y. and Louis, A.A., 2025. Feature learning is decoupled from generalization in high capacity neural networks. HILD at ICML 2025. [10] *N.G. was responsible for the original idea, the majority of the theory and experiments. I aided in the interpretation of theory and data, and was a secondary contributor.*

Mingard, C.*, Ridout, S.*, Grabarczyk R., London, C., Dingle, K., Nemenman, Ilya and Louis, A A. Successful high capacity machine learning models exhibit a universal Zipf's law. Unpublished (to be submitted in late 2025). [11]. *Project conception by C.M., A.A.L. C.M. is primarily responsible for most of the experiments, and theory around PAC-Bayes and Effective Priors. S.R. is primarily responsible for the lower bounds.*

† Göring N., Mingard, C., Nam, Y., and Louis, A A. A simple mean field model of feature learning. Submitted to ICLR 2026. [12]. *N.G. was responsible for the original idea, the majority of the theory and experiments. I aided in the interpretation of theory and data, and was a secondary contributor.*

* denotes joint first author.

† denotes papers that are not submitted as part of this thesis, but are related and may be referenced for support or further commentary

Note: Figs. 6.3, 6.4 and 6.6 are also found in the thesis of Yoonsoo Nam (also found in [8]). They were created by Y.N. (with input from C.M.), and Fig. 6.5 used data from both Y.N. and C.M.

Contents

1 Introduction and Motivation **1**

2 Background & Notation **4**

2.1 Supervised learning 4
2.2 Bayesian Learning Agents 6
2.3 Solomonoff Induction & Kolmogorov Complexity 8
2.4 Linear Models and Kernel Methods 10
2.5 Neural Networks 13
2.6 Parameterisations 16
2.6.1 A Worked Example: 2-Layer Linear Network 17
2.6.2 The Role of Alignment in Update Scaling 21
2.7 Kernel limits 22
2.7.1 The Standard Parameterisation (NNGP limit) 22
2.7.2 The NTK limit 23
2.7.3 The feature learning limits 24

I A toy model of generalisation in neural networks **25**

3 Neural networks learn boolean functions with short DNFs **26**

3.1 Preliminaries and intuition 28
3.1.1 Boolean functions and their disjunctive normal form 29
3.1.2 DFCN–DNF correspondence 30
3.2 Untrained neural networks 33
3.2.1 A DFCN induced prior over Boolean functions 33
3.2.2 Simplicity bias in the prior 33
3.2.3 Understanding P(f) vs K(f) 34
3.2.4 Understanding $P(f)$ in terms of the Universal Prior 36
3.3 Trained neural networks 36
3.3.1 Training algorithms 36
3.3.2 Weight decay adds an additional bias in the posterior 37
3.3.3 The special case of parity 39
3.4 Remarks 40

II Generalisation in Neural Networks & Kernel Methods 43

4 The chaotic regime 44
4.1 Quantifying inductive bias with Bayesian priors 46
4.1.1 Priors over complexity 48
4.1.2 Artificially restricting model capacity 49
4.2 Calculating the Bayesian posterior and likelihood 51
4.2.1 Beyond the Boolean model: MNIST & CIFAR-10 55
4.3 Remarks 56

5 Zipf's law 59
5.1 A universal Zipf's law in the initialisation prior 62
5.2 Where does Zipf's law come from? 63
5.3 Zipfian priors are necessary for good learning performance 66
5.3.1 Existing bounds fail to guarantee good generalisation for subzipfian priors 67
5.3.2 A subzipfian prior guarantees nonzero generalisation error . 67
5.3.3 A superzipfian prior underfits many target functions 68
5.3.4 The Solomonoff–Levin prior is asymptotically Zipfian 69
5.4 "Effective priors" for SGD are often Zipfian 72
5.5 Non-Zipfian learning machines exist and show inferior performance 72
5.6 Remarks 73

III Feature learning & Optimisation 75

6 Feature Learning 76
6.1 Is SGD a Bayesian Sampler? Well, almost 77
6.1.1 Remarks 81
6.2 Is SGD a Bayesian sampler? Depends on the neural network 81
6.2.1 Feature learning 81
6.2.2 Not actual feature learning 82
6.2.3 Remarks 84
6.3 Feature Learning: A Framework 84
6.3.1 Features from the eigenfunctions of forward feature map . . 87
6.3.2 Measuring feature learning by projecting onto the learned function and the target function 88
6.3.3 Eigenvalues and effective dimensions 91
6.3.4 The minimum feature (MF) regime and the extended feature (EF) regime 92

6.3.5 Architectures and Datasets used in experiments 94
6.4 Feature Learning: Results 95
6.5 Remarks 103
6.5.1 Near the no-feature-learning regime 103
6.5.2 In the feature learning regime 104
6.5.3 Parameterisation and Feature Learning 106
6.5.4 Towards a model for feature learning 108

7 Automatic Gradient Descent **111**
7.1 Theory: Parameterisation 113
7.2 Theory: Automatic Learning Rate 114
7.2.1 Majorise-Minimise for Generic Learning Problems 116
7.2.2 Majorise-Minimise for Deep Learning Problems 121
7.3 Results 127
7.4 Remarks 132

8 Conclusion **134**
8.1 Part I: A toy model 134
8.2 Part II: Generalisation 136
8.3 Part III: Optimisation 138

Bibliography **141**

Appendices

A Occam's razor, simplicity and generalisation **164**
A.1 Formalising Occam's razor 164
A.1.1 Statements of the razor 164
A.1.2 The coding theorem and Occam's razor 165
A.1.3 No free lunch, even for Solomonoff Induction 168
A.2 The Case for Simplicity in Practice 169
A.3 Sources of simplicity bias: the parameter-function map 170
A.4 Sources of simplicity bias: Double Descent 172
A.4.1 The bias variance tradeoff 172
A.4.2 Double Descent 174
A.5 Measuring Occams' razor with PAC-Bayes 177
A.5.1 PAC-Bayesian bounds from McAllester [185] 177
A.5.2 PAC-Bayesian bounds from Valle-Pérez and Louis [63], Valle-Pérez et al. [64] 178

B A brief review of generalisation in deep neural networks 180
B.1 A preference towards simplicity . . . 182
B.2 Simplifying limits: Kernel Methods . . . 185
B.2.1 Shortcomings of the kernel approximations . . . 190
B.3 Simplifying limits: Infinitesimal GD . . . 190
B.4 Reintroducing SGD . . . 193
B.5 Mechanistic Interpretability . . . 195

C Appendices for Chapter 3 201
C.1 DFCNs vs FCNs . . . 201
C.2 Why does parity generalise badly? . . . 201
C.3 Important differences between $K_{DNF}(f)$ and $K_{LZ}(f)$ . . . 202
C.4 Proofs from Mingard et al. [1] . . . 203
C.5 Approximating $P(f)$ by sampling . . . 205
C.5.1 Finding the optimum width . . . 207
C.6 Results relating $P(f)$ to $K(f)$ . . . 210
C.6.1 Utility lemmas . . . 210
C.6.2 Function class: constant . . . 213
C.6.3 Function class: entropy . . . 215
C.6.4 Function class: parity . . . 218
C.6.5 Function class: sparse . . . 219
C.7 Trained networks . . . 220
C.7.1 Generating datasets . . . 220
C.7.2 Metropolis-Hastings algorithm . . . 221
C.7.3 Min norm Oracle algorithm . . . 222
C.7.4 SGD-like algorithm . . . 224
C.7.5 Relating Algorithm 4 (steepest descent) to Algorithm 2 (Metropolis-Hastings) . . . 226
C.7.6 Continuous networks . . . 227

D On Complexity 228
D.1 Lempel-Ziv complexity . . . 228
D.2 DNF Complexity Measures . . . 230
D.3 Rescaling complexity measures . . . 231
D.4 Zipf's law . . . 234
D.5 What are the takeaways? . . . 234

E Appendices for Chapter 4 **236**
E.1 Definitions & Algorithms . . . 236
E.1.1 The boolean system . . . 236
E.1.2 Deep Neural Network notation . . . 237
E.1.3 Prior over functions, P(f) . . . 238
E.1.4 Prior over LZ complexity, P(K) . . . 238
E.1.5 Obtaining functions with a range of LZ complexity . . . 239
E.1.6 Generalisation error vs LZ complexity . . . 239
E.1.7 Generalisation vs output function LZ complexity . . . 240
E.1.8 Likelihood estimation and Approximate Bayes . . . 241
E.1.9 Bias-Variance Tradeoff Experiments . . . 243
E.1.10 The Gaussian Processes/Linearised Approximation . . . 244
E.1.11 Experimental details for image datasets & CSR . . . 245
E.1.12 Infinite depth chaotic tanh-activated DNNs have a uniform prior . . . 245
E.1.13 PAC-Bayes appied to a uniform learner . . . 246
E.1.14 The unbiased learner . . . 247
E.2 Further experiments . . . 249
E.2.1 Experiments using the unbiased learner . . . 249
E.2.2 Neural networks with a maximum complexity cutoff K_M . . . 251
E.2.3 PAC-Bayes estimates for $n = 5, 7$ . . . 251
E.3 Selected further experiments . . . 252
E.3.1 Extended data for K-learner with PAC and PAC-Bayes bounds 252
E.3.2 Extended figures of priors $\mathbf{P}(\mathbf{K})$ and $\mathbf{P}(\mathbf{f})$ for DNNs . . . 254

F Appendices for Chapter 5 **262**
F.1 Datasets, Architectures and Algorithms . . . 262
F.1.1 Datasets and architectures . . . 262
F.1.2 Generating the initialisation prior . . . 262
F.1.3 Fitting normalization of the rank-ordered plots . . . 264
F.2 Effective priors for SGD &Loss-based likelihoods . . . 265
F.2.1 Linear Models/Kernel Methods as a Case Study . . . 267
F.2.2 Linear models/Kernel methods: empirical results . . . 273
F.2.3 SGD prior . . . 277
F.2.4 Concluding remarks . . . 279
F.3 Effective Prior Theory . . . 280
F.3.1 Nonexponential likelihoods . . . 280
F.3.2 The effective prior . . . 282
F.3.3 Pseudocode for computing the effective SGD prior . . . 286

F.4 Supplementary experiments . . . 287
F.4.1 Robustness of Zipf's law to architecture and dataset choices 288
F.4.2 Robustness of power-law fits to lower cutoff rank . . . 288
F.4.3 Architecture dependence of function ordering . . . 289
F.4.4 Subzipfian priors on boolean data . . . 289
F.5 Derivation of lower bounds on the generalisation error . . . 293
F.6 A super-Zipfian prior cannot learn very many functions . . . 300
F.6.1 Summary of results . . . 300
F.6.2 Rigorous results . . . 302
F.6.3 Superzipf priors increase the training error for a random function 306
F.7 The PAC-Bayesian bound . . . 308
F.8 Mechanistic origins of Zipf's law in the Kernel limit . . . 309
F.8.1 Zipf's law from low-dimensional latent variables . . . 310
F.8.2 Zipf's law from kernels . . . 312
F.8.3 Zipf's law for equicorrelated kernels . . . 319
F.8.4 Experimental results . . . 323
F.8.5 How close are NNGP kernels to equicorrelated? . . . 324
F.8.6 Infinite depth chaotic tanh-activated DNNs have a uniform prior . . . 325
F.9 Structured priors can produce superzipfian power laws . . . 326
F.10 Lemma on sums of kernel eigenvalues . . . 327
F.11 Predictive information and Zipf's law . . . 328
F.11.1 The predictive information . . . 329
F.11.2 Predictive information for power-law distributions in function space . . . 329
F.12 Fat tails: basics of Zipf's law rank plots . . . 330

G Appendices for Chapter 6 **333**
G.1 Approximating the eigenfunctions and eigenvalues in practice . . . 333
G.2 DNN performance for a fixed feature map as a function of width . . . 335
G.3 On the definition of features . . . 337
G.4 Two lemmas . . . 338
G.5 Accuracy of approximation methods . . . 340
G.5.1 The Nyström method . . . 340
G.5.2 Function space approximation . . . 342
G.5.3 Error in projection measures . . . 344
G.6 The first eigenfunction of the $\mathbf{T}_{\mathbf{MP}}$ operator is a constant . . . 346
G.7 Experimental details . . . 347
G.7.1 Architecture details . . . 347
G.7.2 Learning rate and hyperparameters . . . 348
G.7.3 Loss . . . 349

H Appendices for Chapter 7 **350**
H.1 Proofs 350
H.2 PyTorch Implementation 358
H.3 Other optimisers 359

1 Introduction and Motivation

In 2012, a deep neural network (DNN) called AlexNet [13] achieved a top-5 error rate of 15.3% on the ImageNet competition [14], a huge 10.9% point improvement over second place – which was also held by a neural network. This date is often considered the birth of the modern era of deep neural network-based AI. Since then, much research has been devoted to understanding and improving DNNs. Enormous advances in theory, hardware and software have propelled the field towards partially or completely solving problems no other approaches came anywhere near to, including Go [15], Starcraft 2 [16], protein structure prediction [17], and autonomous driving [18]. In 2024, Gemini Flash 2.0 – a large language model (LLM) – scored 89.7% on the challenging MATH benchmark [19]. Even extremely advanced problems are in reach of the very best LLMs as of 2025 [20].

In each case, the following broad strategy was employed. **First**, a large dataset was collected (e.g. proteins) or generated (e.g. through a reinforcement learning agent playing starcraft). This dataset should be sufficiently representative of the task. **Second**, a sufficiently expressive neural network was created using parameterised linear transformations and (typically) non-parameterised non-linearities. The number of parameters can be in the hundreds of billions. **Third**, the neural network is randomly initialised and trained using a stochastic optimiser. At each training step, a loss is computed on a minibatch, the gradient of the loss

is calculated with respect to the parameters, and an optimiser uses this information, plus a swath of hyperparameters, to adjust the weights. This continues until the training loss is sufficiently small.

This process of learning the training data often also leads to excellent performance on unseen test data. Why?

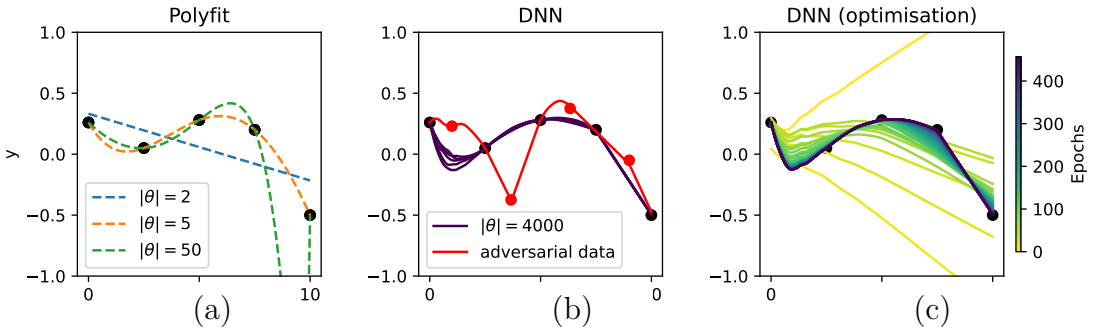


Figure 1.1: Example: How models generalise: capacity, inductive bias, and optimisation. (a) Polynomial regression on $n = 5$ datapoints with parameter counts $|\theta| = 2, 5, 50$: too few parameters underfit, a moderate model $|\theta| = 5$ captures the trend, and an over-parameterised model $|\theta| = 50$ overfits (illustrating classical capacity control). (b) A deep neural network with $|\theta| \approx 4000$ fitted to the same five points consistently learns a smooth interpolating function across runs, yet it can realise a much more complex function when additional "adversarial" points (red) are imposed (evidence of good inductive bias). (c) Training dynamics for the DNN: curves show the function's evolution over epochs from random initialisation to the final fit, highlighting that the optimiser's trajectory selects among many near-zero-loss solutions and therefore may act as a source of inductive bias.

There are essentially only two ways to produce good performance on unseen data. Consider Fig. 1.1, with $n = 5$ datapoints and a range of models with $|\theta|$ parameters.

Capacity control is the classical learning theoretical approach. Fig. 1.1(a) shows how this works on a 1D polynomial fitter. When the number of parameters is just right (in this case, 5), we get a good prediction[1]. When too few ($|\theta| - 2$) the prediction is too simple, and when too large (50) we get overfitting – the model is too expressive, and may pick a very complex function. The sweet spot is when the most complex function expressible by a model with p parameters also happens to be a good description of the data.

[1]We train with noiseless data here; in general the optimum number of parameters will be lower than the number of datapoints

A good inductive bias is the alternative way to make a good learning agent. Fig. 1.1(b) shows how a neural network with 4000 parameters fits a smooth function on 5 datapoints on all of the 5 runs, but could fit a much more complex function, when forced to also fit the adversarial datapoints (datapoints not taken from the target function). The neural network must then (somewhat tautologically) be fitting a simple function because of an inductive bias towards simple functions (for this task).

There is one more question to ask: how do we train the system? Fig. 1.1(c) shows the evolution of the function during the optimisation process. The role and complexity of this process differ drastically between simple models and deep neural networks:

Optimisation for Linear Models For models like polynomial fitting, the optimisation task is straightforward. The Mean Squared Error loss function is convex, which guarantees a single global minimum. This unique solution can often be calculated analytically via a closed-form expression (e.g., the normal equation), completely bypassing the need for iterative optimisation. When an optimiser is used, its primary role is to find this pre-determined solution efficiently; the choice of algorithm affects the speed of convergence, but not the final model itself.

Optimisation for Deep Networks In sharp contrast, the loss landscape for a deep neural network is highly non-convex, containing a vast number of different parameter configurations that achieve minimal training loss. Here, the optimiser's role is elevated: it does not simply find a solution, but actively *selects* one from an ocean of possibilities. The choice of algorithm (e.g., SGD, Adam) and its hyperparameters (learning rate, momentum) dictates the trajectory through the parameter space. This path-finding process itself acts as a powerful inductive bias, able to induce different degrees of *feature learning*, and thus how well the solution generalises.

Thus, understanding the optimiser is not merely a question of computational efficiency, but a fundamental part of understanding deep learning.

2 Background & Notation

2.1 Supervised learning

Supervised learning [21] uses labelled data to learn. Consider an input space $\mathcal{X}$ and output space $\mathcal{Y}$. We can define a measure q from which we can sample input-output pairs $(x, y) \sim q(\mathcal{X}, \mathcal{Y})$. This tuple of $(\mathcal{X}, \mathcal{Y}, q)$ defines a dataset.

For example, MNIST is the dataset 28×28 greyscale images that correspond to the handwritten digits 0-9. In this case, $\mathcal{X} = \mathbb{R}^{28\times 28}$ and $\mathcal{Y} = \{0, \ldots, 9\}$. For this dataset, q exists conceptually but is intractable to specify. Taking a sample $(x, y) \sim q$ (by say, drawing one) would return a handwritten digit and the associated label[1]. Constructing a dataset like this allows for the treatment of ambiguous datapoints and labels.

For supervised learning, we will sample a training set with m examples,

$$S = \{(x_1, y_1), \ldots, (x_m, y_m)\},$$

with each pair independently sampled from q. We will use $S_X = \{x_i\}$ and $S_Y = \{y_i\}$ to denote the inputs and outputs respectively. Given a function $f : \mathcal{X} \to \mathcal{Y}$, its

[1] Practically, there would be a great many subtleties, over for example whether to return ambiguous images, and whether the label space should be a probability distribution over labels or a single label.

projection f_S onto the input data in the training dataset S is denoted

$$f_S = f(S_X) := [f(x_1), \ldots, f(x_m)] \in \mathcal{Y}^m.$$

As discussed in the introduction, the first task of supervised learning is to find a function f that accurately models S – that is, $f_S = S_Y$ (with the aim that f performs well on unseen data). When f is parameterised by θ, we will make this explicit where appropriate, for example, $f(\theta)$, $f(x;\theta)$ or $f_S(\theta)$.

Continuous and discrete outputs We predominantly study c-way classification problems. Neural networks express continuous functions $g(x;\theta)$, where $\mathcal{Y} = \mathbb{R}^c$, which we then threshold to $f(x;\theta) \in \{0, \ldots, c-1\}$.

Note. For classification, most models output a continuous function as an intermediate output (which we denote g), and we use f to denote the thresholded output. When studying regression problems, the continuous output g is the only output.

We measure the degree of accuracy by defining a loss on each pair, $l(g(x_i), y_i)$, where lower indicates greater accuracy. The training loss is $L(g_S) = (1/m)\sum_i l(g(x_i), y_i)$. When performing classification, we often define a separate loss: the train error. The indicator function: $l_{0-1}(f(x_i), y_i) = \mathbb{1}(f(x_i) \neq y_i)$, such that the train error $\epsilon_S = L_{0-1}(f(S_X))$.

However, even if $L(g_S)$ is very low, we have no idea whether it will also be low on new samples. The test loss is given by $L(g_E) = \mathbb{E}_{(x,y)\sim q}[l(g(x), y)]$, and can be estimated with a test set $E = \{(x_1, y_1), \ldots, (x_l, y_l)\}$, randomly sampled i.i.d. from q. Again, for classification, when $L_{0-1}(f_E)$ measures the error rate, the test error is ϵ_G.

Other types of learning While supervised learning relies on explicit input-output pairs, other paradigms exist, such as Reinforcement Learning (RL), where an agent learns to make decisions by interacting with an environment to maximise a cumulative reward signal, without explicit input-output labels [22]. Similarly, Unsupervised Learning aims to find patterns or structures within input data ($\mathcal{X}$) without any corresponding output labels ($\mathcal{Y}$), often used for tasks like clustering

or dimensionality reduction [23]. Semi-supervised learning bridges these, utilising both labelled and unlabelled data for training [24].

2.2 Bayesian Learning Agents

Bayesian inference offers a principled framework for learning functions g (or discrete functions f) given uncertainty. The core idea is to use probability distributions to represent our beliefs about the possible functions that could explain the data.

We can define a **prior distribution over functions**, $P(g)$, which represents our initial beliefs about which functions are plausible before seeing any data. For instance, a prior might favour smoother functions over erratic ones. Given a training set S, we can then update our beliefs using Bayes' theorem to obtain the **posterior distribution over functions**:

$$P(g \mid S) = \frac{P(S \mid g)P(g)}{P(S)},$$

where $P(S \mid g)$ is the **likelihood** of observing the data S given a specific function g, and $P(S) = \int P(S \mid g)P(g)dg$ is the **marginal likelihood** or **evidence**. The integral here is a functional integral over the space of all possible functions, which is typically intractable.

In practice, we often work with functions that are parameterised by a finite set of parameters $\theta \in \Theta$. This simplifies the problem immensely by moving from an infinite-dimensional function space to a finite-dimensional parameter space. The learning process then becomes about inferring the posterior distribution over these parameters.

Let a function be denoted $g(x; \theta)$. The key components of this parameterised Bayesian model are:

1. **The Prior, $P(\theta)$:** A probability distribution over the parameters θ, specified before observing the data. It encodes our assumptions about which parameter values are likely.

2. **The Likelihood, $P(S \mid \theta)$:** This term quantifies how well the model with parameters θ explains the training data S. Assuming the data points in $S = \{(x_i, y_i)\}_{i=1}^m$ are sampled independently and identically (i.i.d.), the likelihood is the product of the probabilities of observing each output given the corresponding input:

$$P(S \mid \theta) = \prod_{i=1}^{m} P(y_i \mid x_i, \theta). \tag{2.1}$$

   The term $P(y_i \mid x_i, \theta)$ is defined by the model's output; for example, for regression with Gaussian noise, it might be a Gaussian distribution centered at $g(x_i; \theta)$.

3. **The Posterior, $P(\theta \mid S)$:** Our updated belief about the parameters after observing the data S. It is calculated using Bayes' theorem:

$$P(\theta \mid S) = \frac{P(S \mid \theta)P(\theta)}{P(S)}. \tag{2.2}$$

4. **The Marginal Likelihood, $P(S)$:** This is the probability of the data integrated over all possible parameter settings, acting as the normalisation constant for the posterior:

$$P(S) = \int_{\Theta} P(S \mid \theta)P(\theta)d\theta. \tag{2.3}$$

   While often difficult to compute, the evidence is crucial for model comparison (a technique known as Bayesian model selection).

Once the posterior $P(\theta \mid S)$ is obtained, it can be used to make predictions for new inputs, averaging over all possible parameter values weighted by their posterior probability. We give an example of a non-parametric Bayesian learning agent in Section 2.3 and a parameterised one in Section 2.4.

2.3 Solomonoff Induction & Kolmogorov Complexity

Key intuitions: 1) The complexity of a string is the length of the shortest program needed to compute it on a universal computing device, meaning that a string consisting of the digits of pi is a simple string, as there is a simple algorithm to produce it. 2) We can use this to define an "optimal" learning agent, by picking the shortest program that produces our training data. See Appendix A for more details.

To introduce Solomonoff induction, we shift our perspective from learning a general mapping $f : \mathcal{X} \to \mathcal{Y}$ to the more fundamental problem of sequence prediction. In this framework, we re-interpret the training data S not as a set of input-output pairs, but as a single, finite binary string that represents an initial sequence, or prefix. For consistency, we can consider this the output sequence S_Y. The learning task is to infer the most probable continuation of this sequence. Accordingly, a hypothesis f is no longer a mapping, but a (potentially infinite) string that has S as its prefix. Solomonoff induction provides a formal, universal solution to this problem by assigning a probability to every possible string continuation.

Kolmogorov Complexity Given some prefix-free Universal Turing Machine U, We define an enumerable semimeasure over strings f based on the probability that U expresses f upon random sampling each bit of a program p until p runs,

$$P(f) = \sum_{p:U(p)=f} 2^{-|p|}, \tag{2.4}$$

where $|p|$ denotes the length of p [25]. $P(f)$ is a semimeasure rather than a full probability measure as some programs may never halt, and thus $\sum_f P(f) \leq 1$. The Kolmogorov Complexity on U of f is given by the length of the shortest program p that runs on U and returns f

$$K(f) = \min\{|p| : U(p) = f\}. \tag{2.5}$$

Kolmogorov Complexities on different UTMs U and V are separated by $O(1)$ constants dependent on U, V but not f such that $|K_U(f) - K_V(f)| < O(1)$ (because

any UTM can simulate any other with a finite program). A result from Levin [26] shows that the prior $P(f)$ can be upper bounded with $K(f)$

$$2^{-K(f)} \leq P(f) < 2^{-K(f)+O(1)}, \tag{2.6}$$

where the $O(1)$ terms can be dependent on U but not on f. The lower bound is straightforward, as the probability of generating the shortest program that produces f is by definition $2^{-K(f)}$.

Solomonoff Induction Within the field of algorithmic information theory, Solomonoff induction is considered the theoretically optimal method for inductive inference, particularly for sequential prediction [27]. Solomonoff's method performs Bayesian inference using the algorithmic prior $P(f)$. Given an initial data sequence S (note S chosen for consistency with training data notation throughout this thesis), the posterior probability of observing a subsequent sequence f is the weighted average of all compatible hypotheses:

$$P(f|S) = \frac{\delta(f_S = S_Y)}{\sum_{f':f'_S=S_Y} P(f')} P(f), \tag{2.7}$$

where $f_S = S_Y$ means "the string f is consistent with S". This provides a formal realisation of Occam's Razor in the form of a Bayesian learning agent, as the prior is dominated by the shortest program, linking it directly to Kolmogorov complexity $K(f)$ via $P(f) \propto 2^{-K(f)}$.

Generalisation The theoretical power of Solomonoff induction lies in its optimality as a universal predictor. If we assume that the data is generated by some computable probability measure, it can be formally shown that the Solomonoff predictor minimises the expected cumulative prediction error. The total expected error is bounded by a term proportional to the Kolmogorov complexity of the true data-generating process [28].

2.4 Linear Models and Kernel Methods

Consider a 1D regression dataset, with $\mathcal{Y} = \mathbb{R}$. We assume the observed targets y_j are generated from an underlying function g^* with additive, independent Gaussian noise: $y_j = g^*(x_j) + \epsilon_j$, where $\epsilon_j \sim \mathcal{N}(0, \zeta^2)$. Here, ζ represents the standard deviation of the irreducible noise in the data. Kernel methods are basic kinds of supervised learning agents [29]. They are made up of 1) a fixed projection φ that maps data $x \in \mathcal{X}$ to an embedding space $\mathcal{E} \subseteq \mathbb{R}^p$ and 2) a $p \times 1$ vector of learnable parameters, to model the continuous function

$$g(x; \theta) = \sum_{i=1}^{p} \theta_i \varphi_i(x) \tag{2.8}$$

where $\varphi_i(x)$ denotes the i'th coordinate of the projection $\varphi(x)$.

Given a training set S, we can reformulate it into a matrix $X_{ij} = \varphi_i(x_j)$ of dimension $m \times p$, consisting of all the inputs $\varphi(x)$, and a target matrix Y of dimension $m \times 1$, consisting of all the targets y. The linear model then attempts to find parameters satisfying $X\theta = Y$.

By basic linear algebra, if $p < m$, the problem may not have an exact solution; if $p = m$, the problem is specified exactly, and if $p > m$, the problem is underdetermined, with many possible solutions. We can find a unique solution by defining a loss function based on the negative log-posterior, combining a data-fit term (from the likelihood) and a regularisation penalty (from the prior). The regularisation strength is controlled by a parameter σ, and the data noise is parameterised by ζ:

$$L(S; \theta) = \frac{1}{2\sigma^2}\|\theta\|^2 + \frac{1}{2\zeta^2}\sum_{j=1}^{m}(y_j - \theta \cdot \varphi(x_j))^2. \tag{2.9}$$

The quantity $\lambda = \zeta/\sigma$ will be useful from here.

The frequentist view We can find parameters $\hat{\theta}$ that minimise the loss over S,

$$\hat{\theta} = (X^T X + \lambda^2 I)^{-1} X^T Y \tag{2.10}$$

where we use the Moore-Penrose inverse when $p > m$ and $\lambda = 0$. This choice of inverse produces the minimum norm solution for $\hat{\theta}$ [30].

Learning with gradient descent (GD) We can find the optimal parameters, $\hat{\theta}$, using infinitesimal gradient descent (gradient flow):

$$\frac{d\theta}{dt} = -\eta \nabla_\theta L(S;\theta),$$

where η is the learning rate, with parameters initialised at $\theta = 0$. This is a convex optimisation problem, so the limit of the GD algorithm (gradient flow) is $\hat{\theta}$ [31]. This method will be more memory efficient than the matrix inversion. Furthermore, studying the effects of gradient descent on linear models is an important first step to understanding the effects of optimisers on non-linear neural networks.

The Bayesian view We can interpret this loss function as the negative log-posterior probability of the parameters. This stems from a Gaussian likelihood function reflecting the observation noise model ($y_j \sim \mathcal{N}(g(x_j), \zeta^2)$), and a zero-mean Gaussian prior on the weights, $p(\theta) = \mathcal{N}(0, \sigma^2 I)$.

$$P(\theta \mid S) \propto \underbrace{\exp\left(-\frac{1}{2\sigma^2}\|\theta\|^2\right)}_{\text{prior}} \underbrace{\exp\left(-\frac{1}{2\zeta^2}\sum_{j=1}^{m}(y_j - \theta \cdot \varphi(x_j))^2\right)}_{\text{likelihood}}, \tag{2.11}$$

$P(S)$ is defined in Eq. (2.3). This is an example of a *parameterised Bayesian learning agent*. Note that the maximum a posteriori in this case is the same as minimising the regularised loss to obtain $\hat{\theta}$ (Eq. (2.10)).

Learning with Kernel Methods The kernel function is defined as the inner product in the feature space, $k(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$. Using this definition, the prediction for a new point x_* in the regularised model can be expressed entirely in terms of the kernel, without ever explicitly computing φ:

$$f(x_*) = k(x_*, S_X)^T (K + \lambda^2 I)^{-1} Y, \tag{2.12}$$

where $K = XX^T$ and $k(x_*, S_X)$ is the vector of kernel evaluations between the test point and the training data.

We can use the so-called kernel trick to learn non-parametrically. A zero-mean Gaussian prior on the parameters, $p(\theta) = \mathcal{N}(0, \sigma^2 I)$, directly induces a zero-mean Gaussian Process (GP) prior over the function space,

$$f(x) \sim \mathcal{GP}\left(0, \sigma^2 k(x, x')\right), \tag{2.13}$$

where the kernel $k(x, x') = \varphi(x)^\top \varphi(x')$ is determined by the parameter covariance.

Given our prior over functions g, we can minimise $L(S; \theta)$ in yet another way: conditioning the GP prior on the data (marginalising over parameters). Once again, this is equivalent to minimising loss (Eq. (2.9)), and produces the same posterior as Eq. (2.11)

The posterior distribution remains a GP. Its mean is given by:

$$\mu_{\text{post}}(x_*) = k(x_*, S_X)^\top (K + \lambda^2 I)^{-1} Y \tag{2.14}$$

This posterior mean is identical to the kernel ridge regression solution (Equation 2.12). It is also the prediction that results from using the maximum a posteriori parameters $\hat{\theta}$ (Equation 2.10), as the MAP estimate for θ is found by minimising the same regularised loss function. The Bayesian approach, therefore, recovers the point-estimate solution as its predictive mean, while additionally providing the posterior variance,

$$\sigma^2_{\text{post}}(x_*) = \sigma^2 \left(k(x_*, x_*) - k(x_*, S_X)^\top (K + \lambda^2 I)^{-1} k(S_X, x_*)\right), \tag{2.15}$$

to quantify predictive uncertainty.

Generalisation The generalisation error of linear models follows a distinct curve as the number of parameters p grows. Let us assume the true signal in the data lies in rank $a \leq m$ feature dimensions. When the model is under-parameterised ($p \ll a$), it lacks the capacity to capture this signal, resulting in high error due to high bias. As p approaches a, the model enters the classical regime, achieving low test error by fitting the signal without overfitting noise. However, at the interpolation threshold ($p = m$), the model has enough capacity to fit every data

point perfectly, including noise, which for large datasets can cause the test error to peak [30]. Finally, in the over-parameterised regime ($p \gg m$), optimisers like Gradient Descent implicitly regularise the solution by prioritising directions of large variance in the data, effectively finding a low-norm solution that fits the signal while ignoring noise, causing the test error to descend again (a phenomenon called double descent – see Appendix A.4.2 for details).

2.5 Neural Networks

Here, we define neural networks, loss functions, stochastic optimisers and regularisation. See [32] for more information.

Deep Neural Networks [32] are parameterised function approximators, constructed by composing multiple layers of affine transformations and element-wise non-linearities. A depth L fully connected network (FCN), with parameters $\theta = \{W^{(l)}, b^{(l)}\}_{l=1}^{L}$, is a function $g(\cdot\,;\theta) : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ defined as the composition of its layers:

$$g(x;\theta) = \zeta \circ (h^{(L)} \circ \psi^{(L-1)} \circ h^{(L-1)} \circ \cdots \circ \psi^{(1)} \circ h^{(1)})(x), \tag{2.16}$$

where $h^{(l)}(z) = W^{(l)}z + b^{(l)}$ is the l'th affine transformation, and $\psi^{(l)}$ is the l'th non-linear activation function. ζ is an optional non-linearity for the last layer, $h^{(L)}$. When the neural network is performing classification, we also introduce $f(x;\theta)$, the class the network outputs. This is obtained by some kind of thresholding function (for binary classification with a single output neron $f(x;\theta) = \mathbb{1}[g(x;\theta) > 0]$, and for multiclass, $f(x;\theta) = \arg\max g(x;\theta)$). By this definition, a network has a depth equal to the number of weight matrices. The number of hidden layers is $L - 1$, so a depth 2 FCN has 2 weight matrices and 1 hidden layer.

The FCN is the "basic" neural network, but many other architectures exist, for example: convolutional Neural Networks (CNNs) [32] for image recognition, or LSTMs [33] and Transformers [34] for language. All share the same basic premise: parameterised linear layers with non-linearities in between.

Activation Functions The non-linear functions φ are crucial for giving the network its expressive power, allowing it to approximate non-linear functions. They are typically applied element-wise to the output of a layer. Two common choices include:

- **Rectified Linear Unit (ReLU):** The ReLU function is defined as $\text{ReLU}(z) = \max(0, z)$. It is computationally efficient and has been shown to work very well in practice, in part due to its ability to produce sparse activations [35].
- **Hyperbolic Tangent (tanh):** The tanh function squashes its input into the range $(-1, 1)$, which can be useful for normalizing activations between layers [32].

Normalisation Layers Normalisation layers primarily serve to stabilise and accelerate the training process. By re-centring and re-scaling the activations of a layer, they help keep training stable, especially at higher learning rates (see the next paragraph on optimisation). Two prominent types are Batch Norm [36] and Layer Norm [34].

Training with gradient-based optimisers Neural network parameters are typically initialised with small random values, see Section 2.6 for the effect of the initialisation scheme. The network is then trained using a variant of Stochastic Gradient Descent (SGD). The core idea of SGD is to iteratively update the parameters θ to minimise a loss function L that measures the discrepancy between the network's predictions $g(x; \theta)$ and the true target values y.

Instead of computing the true gradient over the entire training set S, which would be computationally prohibitive for large datasets, SGD approximates this gradient using a small, randomly sampled subset of the data called a **mini-batch**, $B \subset S$. The parameters are then updated using the gradient of the loss computed on this mini-batch:

$$\theta \leftarrow \theta - \eta \boldsymbol{\nabla}_\theta \left(\frac{1}{|B|} \sum_{(x_i, y_i) \in B} L(g(x_i; \theta), y_i) \right), \tag{2.17}$$

where η is the learning rate: a hyperparameter that controls the step size. While η can be adapted during training, it is typically independent of the parameter values themselves. The gradient $\nabla_\theta L$ is calculated efficiently using the back-propagation algorithm.

There are many variations on SGD. The most popular optimisers include Adam [37], SGD with momentum (or other extras) [38] and LAMB (for large transformers) [39].

Loss Functions The choice of loss function L depends on the nature of the task.

- **Mean Squared Error (mse)** For regression problems, the goal is typically to minimise the mse between the prediction and the true value:

$$L_{\text{mse}}(g(x;\theta), y) = (g(x;\theta) - y)^2.$$

- **Cross-Entropy (CE)** For classification problems, the cross-entropy loss is a standard choice. ζ (the output nonlinearity) is the softmax function for multiclass classification (vector network output) and the sigmoid function for binary classification (scalar network output), such that the output of the neural network is mapped to a probability distribution **Multiclass Classification:** For a multiclass classification problem with C classes, where $\mathbf{y}$ is a one-hot encoded true label vector (i.e., $y_k = 1$ for the true class k and 0 otherwise),

$$L_{\text{CE}}(f(x), y) = -\sum_{k=1}^{C} y_k \log(g(x;\theta)_k).$$

  Binary Classification: For binary classification,

$$L_{\text{CE}}(f(x), y) = -[y \log(g(x)) + (1-y)\log(1-g(x))].$$

Regularisation To prevent overfitting (learning false signals – i.e. noise – in the training data that adversely impacts generalisation), regularisation techniques are employed. A very common method is L2 regularisation, also known as **weight**

decay. This technique adds a penalty term to the loss function proportional to the squared magnitude of the network's weights:

$$L_{\text{total}} = \left(\frac{1}{|B|} \sum_{(x_i, y_i) \in B} L(g(x_i; \theta), y_i) \right) + \lambda \sum_{l=1}^{L} \|W^{(l)}\|_2^2.$$

The hyperparameter λ controls the strength of the regularisation. This penalty discourages large weight values, promoting simpler models that are less likely to overfit. Biases are typically excluded from this penalty. The stochasticity introduced by using mini-batches also provides a form of implicit regularisation. See Section 2.4 for a discussion of L2 regularisation and overfitting in linear models (see also Appendices A and B.2).

2.6 Parameterisations

Here, we summarise the different ways of parameterising a deep neural network, and provide a worked example on a 2-layer linear FCN to explain how the parameterisation leads to different dynamics (based on a shorter version of this argument in [40]). See Section 2.5 for neural networks. See [41, 40] for more details.

The choice of parameterisation for a neural network influences the training dynamics, especially for networks with very wide layers. We consider a fully connected network (FCN) of depth L, input dimension d_{in}, and hidden width n. The weights of each layer, $W^{(l)}$, are parameterized as $W^{(l)} \to a_l W^{(l)}$, where a_l is a deterministic scaling factor and the entries of $W^{(l)}$ are trainable parameters, typically initialized as i.i.d. samples from a standard normal distribution $\mathcal{N}(0, 1)$. For simplicity, we assume all biases are zero.

For a depth-2 FCN (equivalently, a 1-hidden-layer FCN), the output function takes the form:

$$g(x; \theta) = a_2 W^{(2)} \psi(a_1 W^{(1)} x) \tag{2.18}$$

Here, $W^{(1)}$ and $W^{(2)}$ are the weight matrices with entries drawn from $\mathcal{N}(0, \sigma_1^2)$ and $\mathcal{N}(0, \sigma_2^2)$ respectively, and ψ is a nonlinear activation function. The scaling

factors, a_1 and a_2, depend on the chosen parameterisation scheme. The standard parameterisation (SP), for instance, sets $a_1 = a_2 = 1$.

Different parameterisations, such as the Neural Tangent Kernel (NTK) and Maximum Update Parameterisation (μP), apply distinct scaling to the weights and learning rates, leading to different behaviours as the network width $n \to \infty$. μP was introduced by Yang et al. [42] as the update scaling designed to lead to the greatest feature learning in all layers, and for optimiser hyperparameters to be fixed as a function of network width.

The table below summarises the scaling factors for several common schemes.

		Init. scale	Param. multiplier	Gradient ($\times\eta$)
SP	Input	1	1	n^{-1}
	Inter	$n^{-1/2}$	1	n^{-1}
	Readout	$n^{-1/2}$	1	n^{-1}
NTK	Input	1	1	1
	Inter	1	$n^{-1/2}$	1
	Readout	1	$n^{-1/2}$	1
μP	Input	1	1	n
	Inter	$n^{-1/2}$	1	1
	Readout	n^{-1}	1	n^{-1}

Table 2.1: Parameterisation scales. Init. scale denotes the standard deviation of the weights.

2.6.1 A Worked Example: 2-Layer Linear Network

In this section, we discuss the link between parameterisation and feature-learning in the infinite width limit. For a broader discussion on the dynamical effects that lead to feature learning, see Nam et al. [43]. To build intuition, we analyse a 2-layer linear network with a scalar input $x \in \mathbb{R}$:

$$g(x; U, V) = aVUx \tag{2.19}$$

where U is the first-layer weight matrix of dimension $(n \times 1)$ and V is the second-layer weight matrix of dimension $(1 \times n)$. The scaling factor $a = a_U a_V$ encapsulates the parameterisation-specific scalings (as the network is linear, they can be combined

into an overall scale factor). We consider a single gradient update step for a data point (x, y) using the Mean Squared Error loss, $\mathcal{L} = \frac{1}{2}(g(x) - y)^2$.

The gradients with respect to the weights are:

$$\frac{\partial \mathcal{L}}{\partial V} = (g(x) - y)aU^T x \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial U} = (g(x) - y)aV^T x$$

Dropping the common factor $(g(x) - y)$ for clarity, the update directions are proportional to U^T for V and V^T for U. Let the learning rates for each layer be $\eta^{(V)}$ and $\eta^{(U)}$. The updated weights, V' and U', are:

$$V' = V - a\eta^{(V)}U^T \tag{2.20}$$

$$U' = U - a\eta^{(U)}V^T \tag{2.21}$$

The function after one update step becomes:

$$\begin{aligned} g'(x) &= aV'U'x \\ &= a(V - a\eta^{(V)}U^T)(U - a\eta^{(U)}V^T)x \\ &= aVUx + a^2\left(-\eta^{(V)}U^TU - \eta^{(U)}VV^T + a\eta^{(V)}\eta^{(U)}VU\right)x \end{aligned} \tag{2.22}$$

Let's now examine the magnitude of the update terms as $n \to \infty$ for different parameterisations. We initialize entries of U and V from $\mathcal{N}(0, 1)$.

Standard Parameterisation (SP) In SP, the scalings are $a_V = 1$ and $a_U = 1$, with learning rates $\eta^{(V)} = \eta^{(U)} = \eta n^{-1}$. The initial weights are drawn from $V \sim \mathcal{N}(0, n^{-1})$ and $U \sim \mathcal{N}(0, 1)$. Consequently, the vector norms scale as $\|V\|_2^2 = O(1)$ and $\|U\|_2^2 = O(n)$. Since the weights are i.i.d., the magnitude of the network output is $O(1)$, and the norm of the intermediate activation Ux is $O(n^{1/2})$. The layers are updated as follows:

- $\Delta U = -\eta^{(U)}V^T$. Each of its components is of order $O(n^{-1}) \times O(n^{-1/2}) = O(n^{-3/2})$. As the weights in U are $O(1)$, this update is vanishingly small, meaning the features of the first layer are effectively frozen.

- $\Delta V = -\eta^{(V)} U^T$. Each of its components is of order $O(n^{-1}) \times O(1) = O(n^{-1})$. Since the weights in V have a magnitude of $O(n^{-1/2})$, this update is non-trivial and allows the final layer to train.

The output of the network in Eq. (2.22) changes with

- $\eta^{(V)} U^T U$: The term $U^T U = \|U\|_2^2$ is $O(n)$. With $\eta^{(V)} = O(n^{-1})$, this product is $O(1)$.

- $\eta^{(U)} V V^T$: The term $V V^T = \|V\|_2^2$ is $O(1)$. With $\eta^{(U)} = O(n^{-1})$, this product is $O(n^{-1})$.

As $n \to \infty$, the update is dominated by the first term, $-\eta^{(V)} U^T U$. This corresponds to a change in the network's function that is linear in the initial features U, a characteristic of "lazy training" [44]. The network behaves like a kernel machine with the Neural Network Gaussian Process (NNGP) kernel, where only the last layer effectively trains.

NTK Parameterisation. In the NTK parameterisation [45], $a_V = n^{-1/2}$ and $a_U = 1$, learning rates $\eta^{(V)} = \eta^{(U)} = \eta$ and weights initialised as $V_i \sim \mathcal{N}(0, 1)$ and $U_i \sim \mathcal{N}(0, 1)$. This leads to initial vector norms of $\|V\|_2^2 = O(n)$ and $\|U\|_2^2 = O(n)$. With this setup, the function output $g = V n^{-1/2} U x$ has a magnitude of $O(1)$ at initialization (as U and V are uncorrelated the CLT applies), while the intermediate activation norm is $\|Ux\|_2 = O(n^{1/2})$. Let's analyse the update to each layer.

- The update to the first layer, $\Delta U \propto -\eta n^{-1/2} V^T$, has a vector norm of $\|\Delta U\|_2 \propto O(n^{-1/2}) \|V\|_2 = O(n^{-1/2}) O(n^{1/2}) = O(1)$. The parameter vector U moves by a finite amount during training.

- The update to the second layer, $\Delta V \propto -\eta n^{-1/2} U^T$, has a vector norm of $\|\Delta V\|_2 \propto O(n^{-1/2}) \|U\|_2 = O(n^{-1/2}) O(n^{1/2}) = O(1)$. This layer's parameter vector also moves by a finite amount.

Although the parameters undergo a non-vanishing update (i.e., they are not "frozen"), the update norm of $O(1)$ is small relative to the initial parameter norm of $O(n^{1/2})$. This is characteristic of the lazy regime, where parameters do not stray far from their initialisation. Now, we analyse the update to the entire function f. The change Δf is proportional to $\left(n^{-1/2}\right)^2 (-\eta^{(V)} U^T U - \eta^{(U)} V V^T)$.

- The term from the first layer's gradient is $n^{-1}\eta^{(V)} U^T U$. With $\eta^{(V)} = O(1)$ and $U^T U = O(n)$, this term contributes $O(1)$ to the function change.
- The term from the second layer's gradient is $n^{-1}\eta^{(U)} V V^T$. With $\eta^{(U)} = O(1)$ and $V V^T = O(n)$, this term also contributes $O(1)$ to the function change.

Both terms are of the same large order, meaning both layers contribute equally and significantly to the evolution of the function. This is the hallmark of the NTK regime: although individual parameters move only slightly relative to their initial vector norm, their collective movement produces a well-behaved linear evolution of the function, governed by a kernel that remains effectively constant throughout training.

μ-Parameterisation (μP) There are many equivalent ways of parameterising μP [40]. We use scalings $a_V = 1$ and $a_U = 1$, with weights initialized as $U_i \sim \mathcal{N}(0, 1)$ and $V_i \sim \mathcal{N}(0, n^{-2})$. This gives initial norms of $\|U\|_2^2 = O(n)$ and $\|V\|_2^2 = O(n^{-1})$. The learning rates are scaled oppositely to the weight variance: $\eta^{(U)} = \eta n$ and $\eta^{(V)} = \eta n^{-1}$. At initialisation, the network output $g = VUx$ vanishes with a magnitude of $O(n^{-1/2})$. Each layer is updated by

- $\Delta U = -\eta^{(U)} V^T$, has a vector norm $\|\Delta U\|_2 = O(n)\|V\|_2 = O(n)O(n^{-1/2}) = O(n^{1/2})$. This update is of the same order as the weight vector norm $\|U\|_2 = O(n^{1/2})$, indicating that the first layer's features are actively learning.
- $\Delta V = -\eta^{(V)} U^T$, has a vector norm $\|\Delta V\|_2 = O(n^{-1})O(n^{1/2}) = O(n^{-1/2})$. This is also of the same order as the layer's weight norm $\|V\|_2 = O(n^{-1/2})$, showing that the second layer trains non-trivially as well.

The change to the entire function is given by

- The first term, $\eta^{(V)}U^TU$, is of order $O(n^{-1}) \times O(n) = O(1)$.
- The second term, $\eta^{(U)}VV^T$, is of order $O(n) \times O(n^{-1}) = O(1)$.

Both terms contributing to the function's evolution are stable and of the same order of magnitude. This balanced scaling allows the network to learn features at all layers by ensuring that updates are dynamically significant across the network's width. This contrasts sharply with the "lazy" regimes where some layers are effectively frozen.

A key benefit of μP is the ability to perform hyperparameter transfer. Because the dependence on width n is explicitly captured by the parameterisation, optimal hyperparameters found on small models, such as the base learning rate η, were shown empirically to remain optimal for large models, avoiding the need for expensive tuning at scale [46].

2.6.2 The Role of Alignment in Update Scaling

While the analysis in Section 2.6.1 correctly shows that μP balances the *magnitude* of updates, recent work by Everett et al. [47] highlights that this derivation rests on a crucial, implicit assumption: **maximal alignment**. The true effectiveness of training depends not just on the size of a weight update, but on its geometric alignment with other network components.

The calculations in the previous part of this section assumed we aim to balance the norm in the change of the weights, e.g. ΔU should be of a particular order of n. However, what we actually care about is the norm of the weights after updating, $U + \Delta U$. Consider the change in weights, $\|U + \Delta U\|^2 - \|U\|^2 = 2U \cdot \Delta U + \|\Delta U\|^2$. The dot product term scales as $O(\sqrt{n})$ when U and ΔU are uncorrelated (as we would expect at initialisation), but $O(n)$ when they are strongly correlated (towards the end of training). If we assume the important quantity is the change in the norm of the weights, rather than the norm of the update itself, then this would lead to two very different prescriptions for learning rate, as a function of alignment.

For μP trained with SGD, Table 1 in Everett et al. [47] states that at initialisation, the learning rate of the hidden layers $W^{(l)}$ should be $\sqrt{n}$ larger at initialisation than when perfect alignment has been reached.

Everett et al. [47] go on to show that standard parameterisation can outperform μP and perform hyperparameter transfer. However, there is a significant caveat: they redefine Standard Parameterisation to allow for per-layer learning rate scaling, moving it out of the kernel limit.

2.7 Kernel limits

In this section, we explain how different parameterisations lead to closed-form solutions when training networks in the infinite width limit. See Section 2.6 for a description of the parameterisations. See Section 2.4 for a summary of kernel methods. For further reading, see [48] for the NNGP limit, [45] for the NTK limit and [40] for a combined perspective on these limits and the feature learning limits.

The infinite-width limit of neural networks involves taking the width of all hidden layers (i.e. not the input and output layers) to infinity. See Yang et al. and Neal et al. [49, 48] for the precise nature of the limit.

2.7.1 The Standard Parameterisation (NNGP limit)

The Neural Network Gaussian Process (NNGP) limit can be understood in three key ways. It is the limiting behaviour of an infinitely wide neural network parameterised with standard parameterisation.

The prior over parameters picture and most straightforward, is a prior over functions. Sampling parameters θ from an initialisation distribution with SP scaling, $P(\theta) = \mathcal{N}(0, \sigma_w^2)$, induces a prior distribution over functions that is equivalent to sampling from a GP (Eq. (2.13)) with kernel

$$K_{\text{NNGP}}(x, x') = \mathbb{E}_{\theta \sim P(\theta)}[g(x; \theta) g(x'; \theta)]. \tag{2.23}$$

Now that we have this prior, we can perform inference by marginalising over the training data. However, as we saw in Section 2.4, this is the same as

minimising a mse loss function with L2 regularisation with strength ζ^2 and initialisation σ^2 (Eq. (2.9)).

Training the network We can train the DNN with infinitesimal GD (provided certain conditions are met, see [50, 41]).

Frozen network Alternatively, we can decompose the neural network into the last layer θ and treat all hidden layers as a fixed non-linear transform φ [8, 51]. For an FCN, Eq. (2.16), $\theta = W^{(l)}$ and $\varphi = \psi^{(L-1)} \circ h^{((L-1))} \circ \cdots \circ \psi^{(1)} \circ h^{(1)}$. Training this system with continuous GD (as described in Section 2.4) arrives at the same solution.

2.7.2 The NTK limit

The Neural Tangent Kernel (NTK) limit describes the *training dynamics* of an infinitely wide neural network with NTK parameterisation. The key result is that as $n \to \infty$, the network function's evolution over training time t is equivalent to a linear model governed by the neural tangent kernel (NTK), which is fixed in this limit. The function update follows the dynamics of kernel regression [45]:

$$\frac{\partial g(x;\theta_t)}{\partial t} = -\sum_{x',y' \in S} \eta \cdot \Theta(x,x')(g(x';\theta_t) - y')$$

where the NTK, $\Theta(x,x')$, is defined by the inner product of the gradients with respect to the parameters

$$\Theta(x,x') = \nabla_\theta g(x;\theta_0), \nabla_\theta g(x';\theta_0).$$

Lee et al. [52] prove that, in the limit of infinite width, the predictions on test points x_* are given by

$$\mu_t(x_*) = \Theta(x_*,X)\Theta^{-1}(I - e^{-\eta\Theta t})Y \tag{2.24}$$

$$\begin{aligned}\Sigma_t(x_*,x_*) = {} & K(x_*,x_*) + \Theta(x_*,X)\Theta^{-1}(I - e^{-\eta\Theta t})K(X,X)(I - e^{-\eta\Theta t})^T\Theta^{-1}\Theta(X,x_*) \\ & - \left(\Theta(x_*,X)\Theta^{-1}(I - e^{-\eta\Theta t})K(X,x_*) + \text{h.c.}\right),\end{aligned} \tag{2.25}$$

where K is the NNGP kernel.

Although the parameters of all layers undergo a non-trivial movement (as seen in the NTK worked example), their collective effect on the function output is surprisingly simple. The function itself evolves as if it were a simple kernel machine, with its trajectory through function space being linear. Like the NNGP limit, the NTK limit is also a "lazy" training regime, but it is lazy in function space, not parameter space.

2.7.3 The feature learning limits

Unlike the NNGP and NTK limits, μP and other feature-learning limits do not admit a closed-form kernel description, as their hidden representations evolve during training [40]. This distinction highlights the boundary between analytically tractable kernel regimes and the more realistic, but harder to characterise, feature-learning behaviour of practical networks.

Part I

A toy model of generalisation in neural networks

This part presents a toy model of a discrete fully-connected network (DFCN) learning Boolean functions to provide an exact, end-to-end picture of generalisation and optimisation. The core contribution is a proven one-to-one correspondence between the DFCN architecture and Disjunctive Normal Form (DNF) logical expressions. This bijection establishes an interpretable, tunable complexity measure, $K_{DNF}(f)$ (the length of the shortest DNF), which maps directly to the network's minimum weight norm. We show that randomly initialised DFCNs possess a strong inductive bias, where the prior probability over functions, $P(f)$, is exponentially biased towards simple functions with low $K_{DNF}(f)$. Using a Bayesian Monte Carlo training algorithm, we demonstrate how this prior predicts generalisation. Functions with low complexity are learnable, while functions with high $K_{DNF}(f)$ are not. Finally, we show that weight decay sharpens this simplicity bias, acting as an explicit penalty on $K_{DNF}(f)$ that promotes learning the minimum DNF representation – which is equivalent to having better features – and significantly improves generalisation.

3

Neural networks learn boolean functions with short DNFs

In this chapter, we present a toy model that illustrates many of the important concepts related to generalisation in neural networks that we will study throughout this thesis.

Mingard, C., Seier, L., Göring, N., Badelita, A.V., London, C. and Louis, A., 2025. Characterising the Inductive Biases of Neural Networks on Boolean Data. arXiv preprint arXiv:2505.24060. [9]

Most work towards understanding neural network generalisation can broadly be organised into three categories (see Appendix B for more details).

1. **Kernel-based theories** usually treat a network's training dynamics in the infinite-width limit as linearised gradient descent, most prominently through the Neural Tangent Kernel (NTK) [45, 53]. These analyses show that the network first fits functions aligned with the top eigenfunctions of the kernel [54]. However, this framework cannot capture feature learning as weights do not evolve during training.

2. **Finite-width feature-learning** analyses the training dynamics but mostly in mean-field settings or for linear neural networks for specific data distributions [44, 55, 56].

3. **Mechanistic interpretability** reveals meaningful structures in trained networks like feature detectors in vision models [57] or induction heads in language models [58]. It identifies what representations emerge without explaining why or how these features develop through the interplay of architecture, data, and training dynamics.

These approaches leave us without an end-to-end, first-principles understanding of generalisation. To bridge this gap, this chapter deliberately steps back from the complexity of modern large-scale models. We introduce a simple toy model where the entire process—from inductive bias to feature learning—can be tracked analytically. Concretely, we study *depth-2 discrete fully-connected networks* (DFCNs) on Boolean functions and show how they provide a complete picture of generalisation in a tractable setting. Our contributions are

1. **Interpretable complexity measure:** We prove a one-to-one correspondence between DFCNs and disjunctive normal forms (DNFs) (Proposition 3.1.1). This allows us to translate geometric notions such as weight norm into a function-level complexity measure $K_{DNF}(f)$.

2. **Analytic characterization of implicit bias:** Randomly initialised DFCNs induce a tractable prior $P(f)$ over Boolean functions. We derive bounds on $P(f)$ in terms of $K_{DNF}(f)$ that show strong simplicity bias in a range of function families (Table 3.2).

3. **Feature learning under a Bayesian lens:** Using both Metropolis sampling and a greedy stochastic gradient descent (SGD)-like algorithm, we demonstrate that generalisation correlates with the function's prior probability $P(f)$. Functions occupying larger volumes in parameter space (which have low DNF complexity) have low sample complexity, while those with small prior probability (like parity) are unlearnable – more data *harms* generalisation.

4. **Weight decay induces stronger simplicity bias and improves feature learning:** For DFCNs, L_2-regularisation translates into a simple multiplicative

factor $e^{-\lambda K_{DNF}(f)}$ in the posterior (3.8). This lets us quantify how weight decay sharpens the native simplicity bias in $P(f)$. This improves generalisation on "easy" targets (small $K_{DNF}(f)$) but not on inherently complex ones (e.g. high-order parity). We also quantify how this optimiser-induced bias can lead to learning better features.

3.1 Preliminaries and intuition

In Section 3.1.1, we introduce Boolean functions along with two canonical representations: the string-representation and the DNF-representation. In Section 3.1.2 we introduce a novel DFCN-representation, which provides a one-to-one correspondence between Boolean functions and DFCNs.

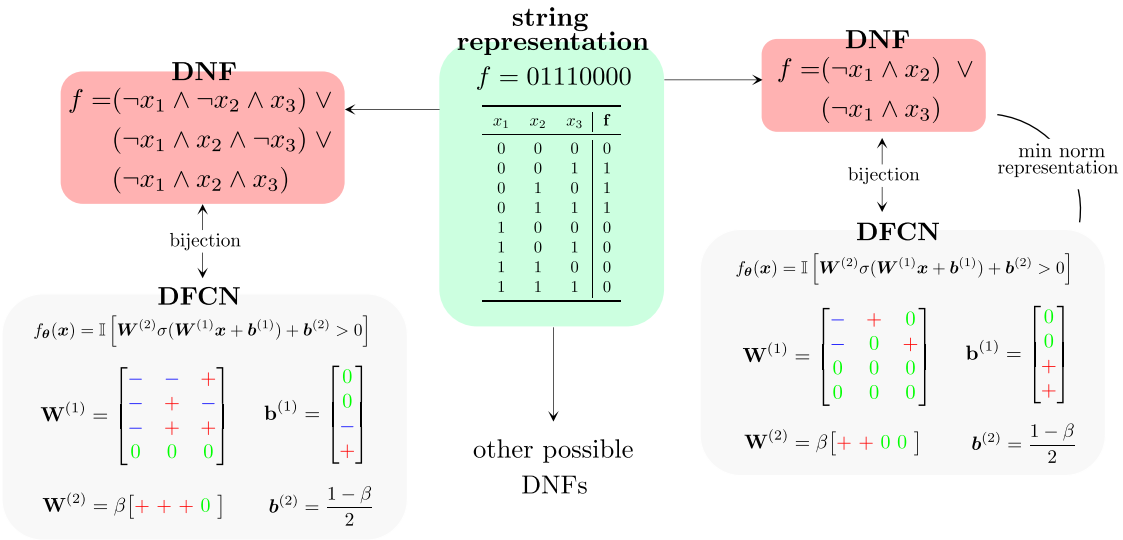


Figure 3.1: Representing Boolean functions Here we show the three ways of representing f. The **green** panel shows the string representation and truth table. The **left red** panel shows how we can extract the DNF representation from the truth table. The **right red** panel shows the minimum DNF representation of f – when the complexity $K_{DNF}(f)$ is minimised. The **grey** panels show how we can represent f by copying the clauses from the **red** panels into a DFCN (with '+'s meaning 1 and '-'s meaning -1). $W^{(1)}$ and $b^{(1)}$ are the weights and biases of the first layer. The combination of the ReLU activation and the bias term (set with Definition 6) ensures that each neuron's output is only 1 when the clause is True, and 0 otherwise. $W^{(2)}$ and $b^{(2)}$ act as the OR operators (plus a global function negation β). The example in the figure uses $\beta = 1$. Note that to guarantee full expressivity, $W^{(1)}$ has dimensions $(2^{n-1} \times n)$.

3.1.1 Boolean functions and their disjunctive normal form

Definition 1 (Boolean function f)**.** *For a fixed input dimension n, a Boolean function maps the 2^n vertices of the n-dimensional hypercube to $\{0,1\}$*

$$f : \{0,1\}^n \to \{0,1\}. \tag{3.1}$$

There are 2^n inputs $x \in \{0,1\}^n$ and 2 outputs $y \in \{0,1\}$ and therefore 2^{2^n} possible functions.

Definition 2 (string-representation)**.** *We define the* string*-representation of a function f as an output string of 0s and 1s where the order is given by an ascending concatenated binary representation of the inputs. See Fig. 3.1.*

There is a second canonical way to represent a Boolean function: with the *disjunctive normal form* O'Donnell [59]. Any Boolean function with n variables can be described by a truth table with 2^n rows (see Fig. 3.1). Each row represents a complete assignment of the variables and therefore corresponds to a conjunction of literals (a clause). To obtain a symbolic description from this table, retain only the rows whose output entry is 1 and take the disjunction (logical OR) of their corresponding clauses. The resulting formula is the function's DNF, defined formally below.

Definition 3 (Literal, clause, DNF)**.** *Let $x \in \{0,1\}^n$ denote a Boolean input vector. A* literal *is either a variable x_i or its negation $\neg x_i$ for some index $i \in [n]$. A* clause *C is a conjunction (AND) of one or more literals: $C(x) = \wedge_{i \in S} v_i,\ S \subseteq [n],\ v_i \in \{x_i, \neg x_i\}$. A Boolean function $f(x)$ can be described by a DNF with t clauses if there exist clauses $C_1, ..., C_t$, and a global negation $\beta \in \{+1, -1\}$ such that*

$$\Phi_f(\mathbf{x}) \ = \ \beta\Big[C_1(\mathbf{x}) \ \vee \ C_2(\mathbf{x}) \ \vee \ \cdots \vee \ C_t(\mathbf{x})\Big].$$

Definition 4 (Length of a DNF)**.** *For a DNF Φ_f whose j-th clause contains $k_j = |S_j|$ literals, its* length *is the total number of literals*

$$L(\Phi_f) \ = \ \sum_{j=1}^{t} k_j. \tag{3.2}$$

When $\beta = -1$, we obtain the logical complement ¬DNF. With this additional global negation, every Boolean function admits a DNF with at most 2^{n-1} clauses [1]. This representation of length $L(\Phi_f) \leq n \cdot 2^{n-1}$ is obtained by enumerating all truth-table rows whose output equals 1 (or 0 when $\beta = -1$, which may be required when $t > 2^{n-1}$); it is called the *canonical expansion* and, up to lexicographic ordering of the clauses, is unique [60, 61]. See Fig. 3.1 for an example. In practice, however, a Boolean function often admits a far shorter DNF Φ_f, and finding such a minimal representation is NP-hard [62]. Conversely, the length can be artificially increased beyond $n \cdot 2^{n-1}$ by padding the formula with tautologically false clauses or duplicating existing ones, yielding DNFs of arbitrarily large length that still compute f. Because raw length can therefore grow without bound, a meaningful complexity measure should refer to the *shortest* realisation of $L(\Phi_f)$ (see Appendix D.2 for further justification).

Definition 5 (DNF complexity)**.** *We thus define the DNF complexity $K_{DNF}(f)$ as the shortest possible DNF expressing f*

$$K_{DNF}(f) \;=\; \min_{\Phi_f} L(\Phi_f). \tag{3.3}$$

3.1.2 DFCN–DNF correspondence

A central concept of this paper is to link the first two well-known representations of a Boolean function to one in terms of DFCNs with a fixed hidden layer width of $\alpha_w 2^{n-1}$, where $\alpha_w \in \mathbb{N}$. A DFCN with this structure is fully expressive [1].

Definition 6 (DFCN)**.** *A DFCN is a depth-two network*

$$f_\theta(\mathbf{x}) = \mathbb{1}\left[W^{(2)}\sigma\Big(W^{(1)}\mathbf{x} + b^{(1)}\Big) + b^{(2)} > 0\right],$$

with ReLU activation σ and the following weight structure:

first-layer weights	$We[1]_{ij}$	$\in$	$\{-1, 0, 1\}$
first-layer bias	$b_i^{(1)}$	$=$	$1 - \sum_j [W_{ij}^{(1)} = +1]$
second-layer weights	$W_i^{(2)}$	$\in$	$\beta \cdot \{0, 1\}$
second-layer bias	$b^{(2)}$	$=$	$(1 - \beta)/2$
global negation	β	$\in$	$\{-1, 1\}$

Table 3.1: DFCN construction.

The central reason we construct a discrete neural network with Definition 6 is that DFCNs are in one-to-one correspondence with a suitable clause-based DNF representation, once we quotient out the obvious redundancies (for example, permutations of hidden units/clauses).

A subtle but important point is that this bijection is *not* a statement that Boolean functions admit unique DNF representations in the usual logical sense. A given function can generally be written by many different DNFs, for example by reordering clauses, adding redundant clauses, or using non-minimal expansions. Rather, the bijection is between our constrained DFCN parameterisation and a corresponding *clause-level DNF representation* modulo the obvious symmetries. In other words, the map is one-to-one at the level of the chosen representation, not at the level of Boolean functions themselves.

Proposition 3.1.1 (DNF-DFCN bijection)**.** *For fixed n, there is a bijection between* (i) *parameter vectors θ satisfying (modulo hidden vector permutations) the restrictions in Table 3.1 and* (ii) *DNF formulas over n variables (allowing for the global β negation).*

Proof. For the proof see Appendix C.4. □

Fig. 3.1 illustrates how the DFCN construction intuitively works. A t-clause DNF can represent any Boolean function with t (or fewer) 1s. Each clause can be represented in a single row of the first layer of the DFCN, with the ReLU activation and correctly set bias term returning 1 if and only if the clause is satisfied. The second layer of 1s and 0s then acts as the OR operator (with a 0 ignoring the clause). The parameter β is there to ensure symmetry between a function and

its complement. As DFCNs and DNFs are in bijection, given a sufficient width, there also exist multiple DFCNs expressing the same Boolean function f. We define $\mathcal{W}_f^{(1)}$ as the set of all matrices $W^{(1)}$ that express f. We now relate the weight norm to the DNF complexity.

Definition 7. *We set the norm of the weight matrices* $(W^{(1)}, W^{(2)})$ *to*

$$\|W\|_2 = \sum_{ij} w_{ij}^2 = \sum_{ij} \mathbb{1}[w_{ij} \neq 0]. \tag{3.4}$$

$\|\theta\|_2 = \left\|W^{(1)}\right\|_2 + \left\|W^{(2)}\right\|_2$ *denotes the overall norm of the DFCN.*

$\left\|W^{(1)}\right\|_2$ corresponds to the number of non-zero entries in $W^{(1)}$, which is equivalent to the number of literals in the DNF representation, allowing us to relate this quantity to $K_{DNF}(f)$.

Proposition 3.1.2. *For* f *represented as a DFCN* f_θ*, the complexity is given by*

$$K_{DNF}(f) = \min_{W^{(1)} \in \mathcal{W}_f^{(1)}} \left\|W^{(1)}\right\|_2. \tag{3.5}$$

Proof. This directly follows from Proposition 3.1.1, see Appendix D.2 for more details. □

The "minimum representation" DFCN (right grey panel of Fig. 3.1) has the lowest $K_{DNF}(f)$; equivalently, $\left\|W^{(1)}\right\|_2$ is minimized. This construction lets us directly link low weight norm to simple functions. Proposition 3.1.1 completes the picture of the following equivalent ways to express any Boolean function f:

- **String representation:** We can represent f using a binary string of output values, ordered by input.
- **DNF representation:** We can represent f using a DNF Φ_f.
- **DFCN representation:** We can represent f by a DFCN of width $\geq 2^{n-1}$.

3.2 Untrained neural networks

In this section, we provide empirical results indicating that individual functions with low DNF complexity occupy a much larger fraction of parameter space than complex functions do. This bias towards simple functions influences training (Section 3.3).

3.2.1 A DFCN induced prior over Boolean functions

Our goal is to understand which Boolean functions a *randomly initialised* DFCN is most likely to compute. Because, by Proposition 3.1.1, a depth-two DFCN is fully specified once its first-layer weight matrix $W^{(1)}$ is fixed, the most agnostic prior is to draw each entry of $W^{(1)}$ independently and uniformly from the ternary set $\{-1, 0, 1\}$. After $W^{(1)}$ is chosen, we flip an unbiased coin for a global sign $\beta \in \{-1, 1\}$. If at least one hidden unit in a row in $W^{(1)}$ is non-zero, we set $W_i^{(2)} = \beta$, else $W_i^{(2)} = 0$. The two bias vectors are deterministic functions of $W^{(1)}$ and β, see Definition 6.

Definition 8 (Prior probability)**.** *Let $\{\boldsymbol{\theta}\}$ denote the finite set of weight vectors θ produced by the sampling procedure above, its size is $|\{\boldsymbol{\theta}\}| = 2 \cdot 3^{n2^{n-1}}$. For a Boolean function f, we define*

$$P(f) \;=\; \frac{\left|\{\boldsymbol{\theta} \in \{\boldsymbol{\theta}\} : f_{\boldsymbol{\theta}} = f\}\right|}{|\{\boldsymbol{\theta}\}|}, \tag{3.6}$$

i.e. the fraction of all admissible parameters that implement f.

Because each weight setting occupies a unit cell of equal size, $P(f)$ is proportional to the volume of weight space assigned to f. Boolean functions f whose string representation can be implemented by many different weight configurations naturally claim a larger share of this volume.

3.2.2 Simplicity bias in $P(f)$

$P(f)$ has emerged as a strong predictor of generalisation [2, 63]. Under a Bayesian update with 01-likelihood on m samples, the posterior weight of any interpolating function f is exactly proportional to $P(f)$. Moreover, the PAC–Bayesian

bound from [64]

$$\epsilon(f) \leq 1 - \exp\left(\frac{-\ln P(f) + \ln(\delta/2m)}{m-1}\right) \tag{3.7}$$

implies that larger $P(f)$ yields tighter expected generalisation error.

Empirical studies showed that the equivalent prior of continuous FCNs is heavily biased toward simple functions. Motivated by general arguments on overparameterised learners from [65], Valle-Pérez et al. [64] observed that $P(f) \lesssim 2^{-K_{\mathrm{LZ}}(f)+\mathcal{O}(1)}$, where $K_{\mathrm{LZ}}(f)$ is the Lempel-Ziv complexity of the string representation of f. While these findings suggested a fundamental connection between function probability and complexity, they relied on a complexity metric that lacks a connection to network architecture. In contrast, DNF complexity $K_{DNF}(f)$ provides a more interpretable measure with explicit ties to the network, as it directly counts the minimal literals needed to express the function (see Appendix C.3 for a further discussion on complexity metrics). This connection to the DFCN allows us to derive analytical bounds on $P(f)$ in the next section.

3.2.3 Understanding $P(f)$ vs. $K_{DNF}(f)$

Fig. 3.2(a) compares the empirical prior probability $P(f)$ obtained from 10^8 i.i.d. parameter draws to the DNF complexity $K_{DNF}(f)$ for $n = 4$ (see Appendix C.5 for full details). Only 631 of the total 65536 functions were not found, indicating that $P(f) \lesssim 10^{-8}$ for these functions. Each datapoint is a function. The minimum complexity constant function (blue) is the most frequent function, with the random t-entropy functions (reds) occupying the upper part of the envelope, and k-parity (greens) the lower. Fig. 3.2(b) shows the dependence of $P(f)$ on n for some function types. k-parity fall much faster with n than the t-entropy with $t = 1, 2, 4$. Can we predict $P(f)$ vs. $K_{DNF}(f)$ at large n?

Theorem 1. *We require $\alpha_w \geq 1$ to satisfy full expressivity. To leading order, for the three function classes defined in this section, $P(f)$ scales as:*

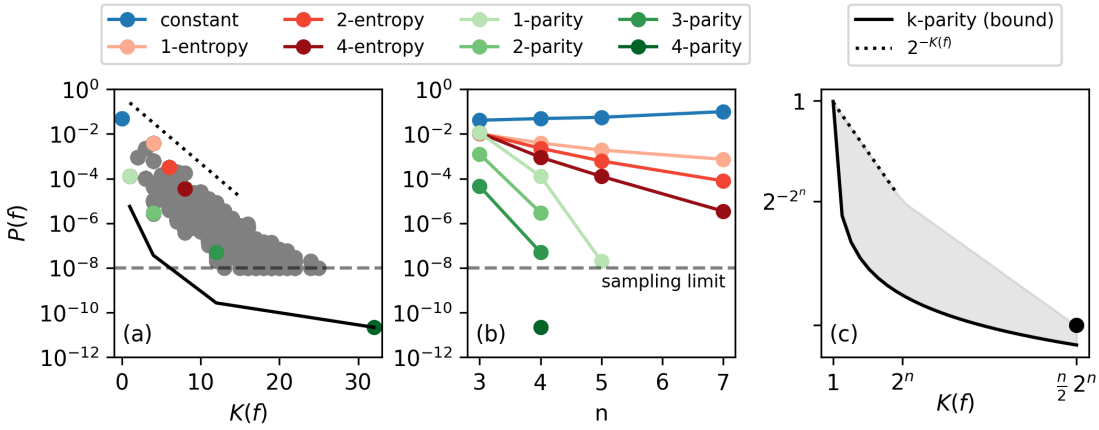


Figure 3.2: Prior probability $P(f)$ vs. DNF complexity $K_{DNF}(f)$ for $n = 4$. The hard cutoff at $P(f) = 10^{-8}$ reflects sampling constraints from 10^8 parameter draws. **(a)** Each point represents a Boolean function, with constant functions (blue, $K = 0$) dominating the parameter space. We use coloured dots to distinguish functions of interest from general functions (grey). Low-complexity functions occupy exponentially larger volumes, with k-parity (greens) suppressed compared to t-entropy (reds) of equal complexity. **(b)** Function probability scaling with input dimension n shows k-parity probability decreasing much faster than t-entropy, matching theoretical bounds in Table 3.2. **(c)** Asymptotic bounds on $P(f)$ for large n values. The black point is parity.

Function class	**Scaling**	**Complexity**	$\log_2(P)/K$ **ratio**
Constant	$0 \leq 1 - P(f) \leq 2^{-O(\alpha_w(4/3)^n)}$	O(1)	$\alpha_w(4/3)^n$
t-entropy $(t = O(n))$	$2^{-O(\alpha_w t(4/3)^n)}$	O(nt)	$\alpha_w(4/3)^n/n$
k-parity	$2^{-\Theta(\alpha_w k 2^{n-1})}$	$k2^{k-1}$	$\alpha_w 2^{k-n}$

Table 3.2: Scaling laws in $P(f)$ as a function of complexity. We only show the leading order terms here, valid when $\alpha_w \gg (3/4)^n$.

Proof. See Appendix C.6.2 for bounds on the constant function, Appendix C.6.3 for t-entropy, Appendix C.6.4 for k-parity. □

We study three canonical families of functions (full descriptions in Appendix C.6).

Constant functions: f outputs the same label on every input – $K_{DNF}(f) = 0$ (minimum representation has $W^{(1)} = 0$).

k-parity: Sparse parity on the first k bits – $K_{DNF}(f) = k2^{k-1}$.

t-entropy: Exactly t ones and $2^n - t$ zeros – $K_{DNF}(f) \leq n \min\left(t, 2^n - t\right)$.

These classes of functions allow us to explore a broad range of complexities. We summarise the most relevant bounds and scaling laws in Table 3.2 (for a full discussion of bounds and assumptions, see Appendix C.6).

Valle-Pérez et al. [64] predicted that $\log_2(P(f)) \leq -aK + b$ for some suitable complexity measure K and constants a, b (independent of K), and empirically observed that for a large class of functions, the bound was very tight. Table 3.2 provides theoretical results showing that for the typical t-entropy function, this scaling is a function not of $K_{DNF}(f)$, but $\alpha_w(4/3)^n/n$. However, for k-parity, $P(f)$ is suppressed by a significantly larger factor: $\alpha_w 2^{k-n}$ (which is dependent on complexity). This extra suppression predicts that $P(f)$ will occupy a wide envelope – with some functions of low complexity but also low probability (a concrete example of predictions is presented in [66]). This is visualised in Fig. 3.2(c).

3.2.4 Understanding $P(f)$ in terms of the Universal Prior

In the background (Section 2.3), we introduced the Universal Prior and Solomonoff induction, as the "gold standard" learning agent. We discuss the connections between $P(f)$ and our computable complexity measures $K_{DNF}(f)$ (and $K_{LZ}(f)$) to Kolmogorov complexity and Solomonoff induction in Appendix D, and in the next part (Chapter 4).

3.3 Trained neural networks

We use $n = 7$. All models are trained on random subsets $S \subset \{0,1\}^n$ of size $m \in \{16, 32, 64, 96\}$, where the rest of the set $\{0,1\}^n$ is used as a test set. Results are averaged over ten independent draws. All runs use the DFCN of Definition 6 with $\alpha_w = 2$ to ensure overparameterisation.

3.3.1 Training algorithms

As DFCNs have a discrete weight space, they cannot be straightforwardly trained using SGD. While SGD variants adapted to discrete weights exist [67], we instead employ a fully Bayesian MCMC algorithm, due to its easier interpretability, and

an oracle algorithm as described below. We also implement a steepest descent random search algorithm. See Appendix C.7 for full details.

Metropolis–Hastings (Algorithm 2). A Metropolis-Hastings algorithm based on the acceptance probability $\alpha = \min\left\{1, e^{-\kappa\Delta\mathcal{L}(\theta)}e^{-\lambda\Delta\|\theta\|_2^2}\right\}$, where Δ denotes the difference between steps, $\mathcal{L}$ is the MSE error, κ is a temperature hyperparameter and λ is the weight–decay coefficient.

Min norm oracle (Algorithm 3). This is an oracle that returns the minimal complexity DNF compatible with the training set S obtained by exhaustive search.

Greedy SGD–like (Algorithm 4). This performs greedy local optimisation. At every step θ_t, we evaluate the loss of every possible neighbour of $(W^{(1)}, W^{(2)})$ with Hamming distance one to θ_{t-1}. We find the set of neighbours which maximally improve the batch error (minimising the loss), picking uniformly from the lowest-norm neighbours with probability p, otherwise picking uniformly from the entire set with probability $1-p$. The hyperparameter p acts like a weight decay parameter: larger p results in a larger bias towards minimum norm functions and hence towards functions of small $K_{DNF}(f)$.

3.3.2 Weight decay adds an additional bias in the posterior

For Algorithm 2 we choose $\kappa = 1000$, which approximates the likelihood term $e^{-\kappa\Delta\mathcal{L}(\theta)}$ in the MCMC sampling as a 01-likelihood $\mathbb{1}[f_\theta(S) = f^*]$. With the uniform prior over θ defined in Definition 8, the posterior over Boolean functions f is obtained by marginalising:

$$P_\lambda(f \mid S) = \frac{\sum_{\{\theta: f_\theta = f\}} \mathbb{1}[f_\theta(S) = f^*] e^{-\lambda\|\theta\|_2^2} P(\theta)}{\sum_\theta \mathbb{1}[f_\theta(S) = f^*] e^{-\lambda\|\theta\|_2^2} P(\theta)} \simeq \frac{e^{-\lambda K_{DNF}(f)} P_{\lambda=0}(f \mid S)}{\mathbb{E}_{f\sim P_{\lambda=0}(\cdot|S)}[e^{-\lambda K_{DNF}(f)}]}, \quad (3.8)$$

where $P_{\lambda=0}(f \mid S)$ is just the posterior induced by a 01-likelihood, $P(f \mid S) \propto \mathbb{1}[f_\theta(S) = f^*]P(f)$. We have assumed that $\sum_{\{\theta: f_\theta = f\}} e^{-\lambda\|\theta\|_2^2}$ is dominated by the smallest attainable norm $\|\theta\|_2^2$ for a given f, and that $\|\theta\|_2^2 \simeq \left\|W^{(1)}\right\|_2^2$, which

gets more accurate for larger n since the parameter space is largely dominated by $W^{(1)}$. As the norm is directly related to the complexity (Proposition 3.1.2), weight decay approximately acts as a multiplicative bias $e^{-\lambda K_{DNF}(f)}$ that further sharpens the simplicity bias in $P(f)$.

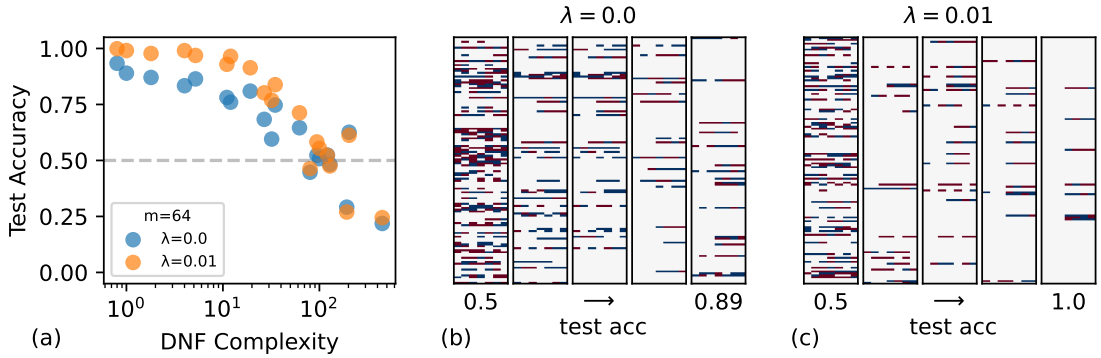


Figure 3.3: Inductive biases of trained DFCNs ($n = 7$) with Algorithm 2 (a) shows how the inductive bias of DFCNs towards lower complexity functions allows them to find such functions more easily than higher complexity functions. It also shows that weight decay increases these biases, being able to achieve 100% test accuracy on some functions. See Appendix C.7 for a list of the functions used. **(b)** and **(c)** show heatmaps of $W^{(1)}$ during training at different test accuracy checkpoints with a target function of 4-parity for no weight decay ($\lambda = 0$) and with weight decay ($\lambda = 0.01$), respectively. Both panels show how the network learns simple representations of the target function, with weight decay managing to learning the exact DNF representation.

The results of training DFCNs with Algorithm 2 are shown in Fig. 3.3. Fig. 3.3(a) shows the inductive bias of the network: higher complexity functions are harder for the network to learn due to the large bias towards low complexity functions. Hence, at zero training error, the network only achieves good test accuracy for simple target functions. Fig. 3.3(b) and Fig. 3.3(c) show heatmaps of $W^{(1)}$ during training at different test accuracy checkpoints for a 4-parity target function. We see that for no weight decay ($\lambda = 0$), the network trains towards a simple representation of the target function (resulting in a sparse $W^{(1)}$ with many zeros), but adding weight decay ($\lambda = 0.01$) greatly improves the generalisation of the network by further increasing the bias towards low complexity functions. At 100% test accuracy (Fig. 3.3(c)), we see that the network has learned the exact minimal DNF representation – i.e. optimal features – of the target function, with 4-parity requiring eight clauses in its minimal DNF.

This gives an intuitive explanation for the general empirical observation that weight-decay improves test performance and the general hypothesis that flatter minima (higher volume or prior weight $P(f)$) generalise better [68, 69, 70].

3.3.3 The special case of parity (Fig. 3.4)

MCMC without weight-decay ($\lambda = 0$). As $\kappa \gg 1$, the posterior is dominated by the prior $P_{\lambda=0}$. Simple parities ($k \leq 4$) are eventually learned, while $k = 6, 7$ never exceed chance level despite more data. Their posterior mass is simply too small for the algorithm to find them within the allotted budget, mirroring the extreme rarity observed in Fig. 3.2. Weight norms stay high and almost k-independent. See Appendix C.2 for a discussion on why more data actually harms generalisation for high-parity target functions.

MCMC with weight-decay ($\lambda > 0$). Penalising the norm dramatically improves generalisation when a low-norm representation exists. For $k = 1$, the network reaches perfect accuracy after $m = 64$ examples, and its weight norm drops to a much lower value than without weight-decay. These performance gains from weight-decay persist up to $k = 4$, beyond that the minimum representation is so large that the decay term can no longer offset the small prior probability (3.8).

Min norm Oracle. The oracle provides the Bayes-optimal accuracy achievable if the learner always selects the lowest-norm DNF that interpolates S. The accuracy gap between the weight-decay sampler and the oracle is small for $k \leq 4$, demonstrating that the algorithm actually discovers the minimal complexity DNF in practice, even though it is capable of expressing much more complicated DNFs for the given string representation.

Greedy SGD-like. We train on parity as well as the other function families with the greedy SGD-like algorithm, see Fig. C.7 in the Appendix. The learning curves look qualitatively similar to MCMC, showing that SGD behaves Bayesian in our DFCN setting, as also claimed in [2].

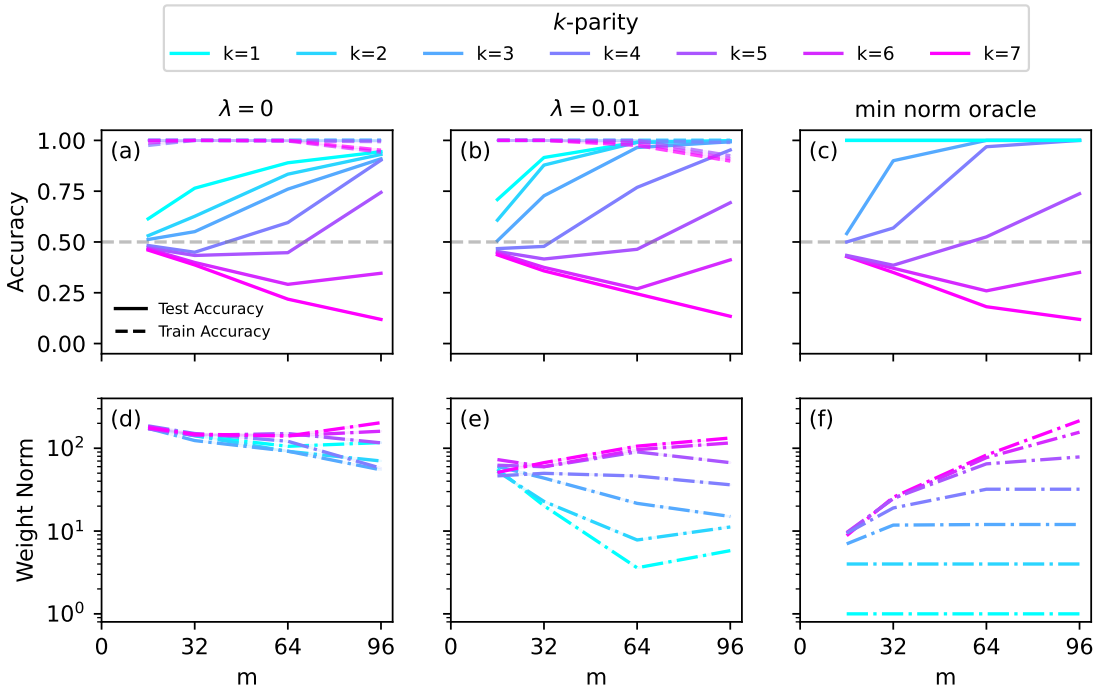


Figure 3.4: Training statistics for k-parity target functions (a), **(b)** and **(c)** show training and test accuracies for various k-parity target functions for the MCMC algorithm (Algorithm 2) without weight decay ($\lambda = 0$), with weight decay ($\lambda = 0.01$) and an oracle algorithm (Algorithm 3), respectively. Weight decay improves test accuracy for all $k < 5$ functions. For 7-parity (the most complex function for an $n = 7$ input DFCN), the model is strongly biased against this function; the more data we give it, the worse the test accuracy will be. **(d)**, **(e)**, and **(f)** show the weight norms for each of the training algorithms, clearly showing that weight decay greatly lowers the norm compared to no weight decay.

3.4 Remarks

State-of-the-art neural networks are so large and the data they ingest so heterogeneous that we can usually only fully understand curated sub-problems – for instance, modular arithmetic [58]. To make causal statements about representation learning, we need a small, exactly-solvable test-bed in which (i) the target function's complexity is tunable, model-independent and human-interpretable, (ii) learned representations live in an easily interpretable space, and (iii) the inductive bias and learned functions can be precisely understood.

Our DFCN offers precisely that – because it represents a DNF, the complexity of the learning problem is controlled directly by the number and size of clauses of the

target function f – the complexity $K_{DNF}(f)$. $K_{DNF}(f)$ is therefore simultaneously meaningful for the data, the hypothesis class, and the parameters. Furthermore, the one-to-one mapping between weights and logical clauses at the level of the constrained construction lets us derive analytic expressions for the prior $P(f)$, turning qualitative notions of "simplicity bias" from [64] into quantitative, testable predictions for generalisation. In this sense, the DFCN plays the same role in deep-learning theory that the Ising model plays in statistical physics: it is the minimal, exactly computable system that still exhibits the phenomena we care about. Sliding along the complexity axis within a single framework demonstrates how complexity, inductive bias, training dynamics, and generalisation interact in real time.

Specifically, we analytically demonstrated that DFCNs with uniform sampled weights (i) induce a strong simplicity bias in the distribution over Boolean function $P(f)$. (ii) This bias in the prior directly determines generalisation with a Bayesian learning algorithm, as best seen for high-parity Boolean functions, where the bias against complex functions is so strong that it cannot overcome the prior, even with additional data points. (iii) Weight decay amplifies this simplicity bias, learning the minimum representation, inducing feature learning. This explains why it improves performance on simple target functions but not on inherently complex ones.

Limitations. Our work focuses on discrete networks with Boolean inputs, which provides analytical tractability but leaves a gap between our theory and typical continuous deep learning applications. The sampling algorithms become computationally intractable for large n, even though this is the interesting regime concerning the bounds in Table 3.2. Furthermore, our training algorithms do not capture all properties of continuous optimisation with SGD or other optimisers like Adam.

Investigating the parameterisation. In the background (Section 2.6), we showed how the choice of initial parameterisation can affect learning (with respect to feature learning). One simple change of parameterisation would be to initialise the last layer with all 0s as opposed to a mix of 0s and 1s. This would likely have a μP-like effect, encouraging feature learning.

Other future directions. One strength of this model is its ability to directly study the effect of optimiser hyperparameters. Exploiting this to study other phenomena observed in continuous neural networks, such as grokking and neural collapse are promising new directions. The most difficult task will be to develop suitable and interpretable optimiser measures that allow for bounds on the prior $P(f)$. This research direction should also better understand how different optimisers and training schemes influence the posterior, or whether a Bayesian formulation is even possible at all.

Part II

Generalisation in Neural Networks & Kernel Methods

This part presents a first-principles explanation of generalisation, stemming from the inductive biases towards simple functions inherent at initialisation. While these biases are easily integrated into the posterior of Bayesian learning algorithms, this isn't the case when using optimisers. We demonstrate that our predictions empirically extend to networks trained with standard optimisers.

In Chapter 4 we analyse the prior of continuous neural networks on the Boolean data. We find that the prior $P(f) \lesssim 2^{-aK(f)}$ can be controllably weakened by tuning the initial weight variance, σ_w to move the network from the ordered regime ($a = 1$) into the chaotic regime (where $a < 1$). When $a < 1$, the prior cannot counteract the 2^K growth in functions, and leads to poor generalisation.

Subsequently, Chapter 5 reveals a universal pattern in this architectural bias: across a wide range of models and datasets, the prior probability of a function $P(f)$ follows Zipf's Law (i.e., $P(f) \propto R(f)^{-1}$). We then prove that this Zipfian prior is not merely an empirical curiosity but a necessary condition for achieving efficient learning.

4

The chaotic regime

In this chapter, we study continuous neural networks on Boolean data. We show that the prior distribution over complexities (1) is strongly biased towards simple functions (2) can predict generalisation, and (3) can be broken by initialising in the chaotic regime.

Mingard C, Rees H, Valle-Pérez G, Louis AA. Deep neural networks have an inbuilt Occam's razor. Nature Communications. 2025 Jan 14;16(1):220 [5].

In Chapter 3, we studied a discrete neural network learning Boolean functions. We used the discretisation to show a one-to-one correspondence between our constrained DFCN construction and DNF representations, up to the natural equivalences discussed there (for example, clause/hidden-unit permutations). This gave us a natural complexity measure for the discrete NN, and allowed us to explore interesting properties of the learning agent: for example, the types of functions that are learned easily, why some simple functions are hard to learn, and what features are learned. We proved some basic results concerning the relation between $P(f)$ and $K_{DNF}(f)$.

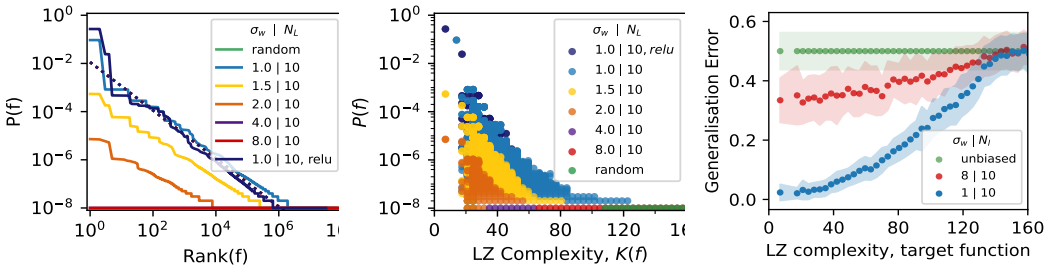
In this chapter, we continue to use the Boolean dataset to probe the prior of DNNs.

1. What does $P(f)$ look like in continuous DNNs?

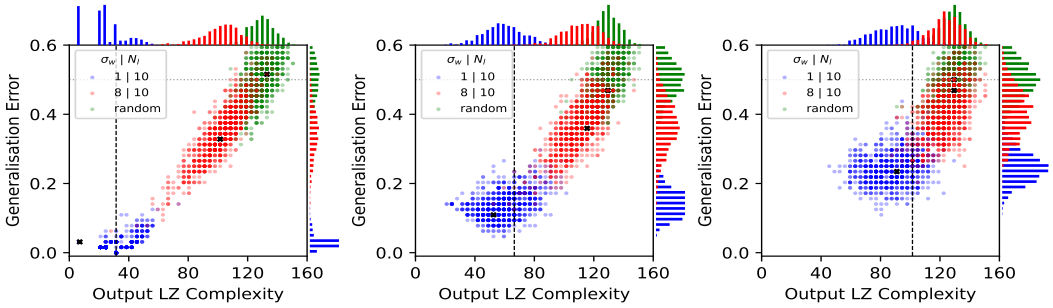
2. How can we break the bias towards simple functions?

3. How strong does the simplicity bias have to be before learning is impossible?

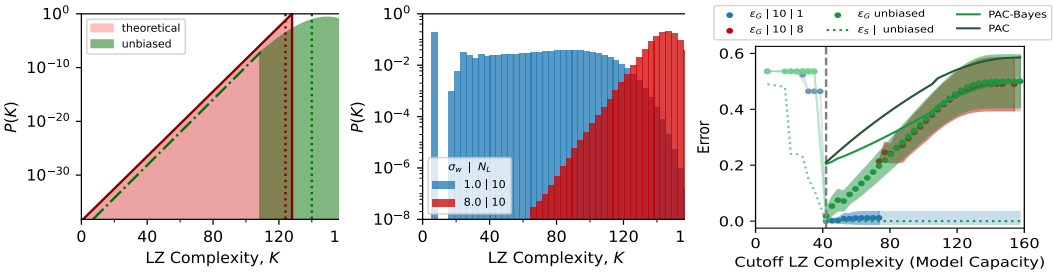
4. Can we abstract out the functions themselves and study these systems at the level of complexity?



(a) Prior $P(f)$ versus rank. **(b)** Prior versus complexity **(c)** generalisation error vs K



(d) Target function LZ = 31.5 **(e)** Target function LZ = 66.5 **(f)** Target fn LZ = 101.5



(g) Prior $P(K)$ **(h)** Prior $P(K)$ for $\sigma_w = 1, 8$ **(i)** Bias variance

Figure 4.1: Priors over functions and LZ complexity (a) Prior $P(f)$ over $n = 7$ Boolean functions from N_l-layer tanh FCNs, sampled 10^8 times with Gaussian weights $\sigma_w = 1 \ldots 8$, compared to ReLU and Zipf's law [64]. (b) $P(f)$ plotted against LZ complexity K. (c) Generalisation error vs. K for tanh networks ($\sigma_w = 1, 8$) and an unbiased learner, trained to zero error on $m = 64$ samples; error computed over $|T| = 64$ (See Fig. E.8 for PAC-Bayes bounds on this data). (d–f) Scatterplots of generalisation error vs. K for three target functions. Dashed line: target K, cross: mode. Histograms show $P_{\text{SGD}}(K|S)$ and $P_{\text{SGD}}(\epsilon_G|S)$. (g) Prior $P(K)$ from random sampling, with 90% of mass right of the dotted lines. Dash-dot: extrapolation. (h) $P(K)$ is flat for $\sigma_w = 1$, biased to high K for $\sigma_w = 8$, explaining performance differences. (i) Generalization error with K-learning for $\sigma_w = 1, 8$ and unbiased learners on $|S| = 100$. No solutions found with low K for $\sigma_w = 8$, or high K for $\sigma_w = 1$.

4.1 Quantifying inductive bias with Bayesian priors

We used the prior over functions, $P(f)$ in the previous chapter (defined in Definition 8 for discrete neural networks). Given a dataset S_X, $P(f)$ is the probability that a DNN $f_\theta(x)$ expresses $f(S_X)$ upon random sampling of parameters over a parameter initialisation distribution $P_{\text{par}}(\theta)$:

$$P(f) = \int \mathbb{1}[f_\theta(S_X) = f] P_{\text{par}}(\theta) d\theta, \tag{4.1}$$

where $\mathbb{1}$ is an indicator function (1 if its argument is true, and 0 otherwise), and $f = f(S_X)$. Explicitly, this term is 1 if the neural network f_θ expresses f with parameters θ, else 0.

It was shown in [64] that, for ReLU activation functions, $P(f)$ for the Boolean system was insensitive to different choices of $P_{\text{par}}(\theta)$, and that it exhibits an exponential bias of the form $P(f) \lesssim 2^{-a\tilde{K}(f)+b}$ towards simple functions with low descriptional complexity $\tilde{K}(f)$, which is a proxy for the true (but uncomputable) Kolmogorov complexity. We will, as in [64], calculate $\tilde{K}(f)$ using $K_{LZ}(f)$, a Lempel-Ziv (LZ) based complexity measure from [65] on the string representation (Definition 2) – the 2^n long bitstring that describes the function, taken on an ordered list of inputs. Other complexity measures give similar results [64, 71], so there is nothing fundamental about this particular choice. To simplify notation, we will use $K_{LZ}(f)$ instead of $\tilde{K}(f)$. The exponential drop of $P(f)$ with $K_{LZ}(f)$ in the map from parameters to functions is consistent with an algorithmic information theory (AIT) coding theorem [28] inspired *simplicity bias* bound [65] which works for a much wider set of input-output maps. It was argued in [64] that if this inductive bias in the priors matches the simplicity of structured data, then it would help explain why DNNs generalise so well. However, the weakness of that work, and related works arguing for such a bias towards simplicity [72, 73, 64, 74, 1, 75, 76, 77, 71, 78], is that it is typically not possible to significantly change this inductive bias towards simplicity, making it hard to conclusively show that it is not some other property

of the network that instead generates the good performance. Here we exploit a particularity of tanh activation functions that enables us to significantly vary the inductive bias of DNNs. In particular, for a Gaussian $P_{\text{par}}(\theta)$ with standard deviation σ_w, it was shown [79, 80] that, as σ_w increases, there is a transition to a chaotic regime. Moreover, it was recently demonstrated that the simplicity bias in $P(f)$ becomes weaker in the chaotic regime [81] (see also Appendix E.1.12). We will exploit this behaviour to systematically vary the inductive bias over functions in the prior.

In Figs. 4.1a and 4.1b we depict prior probabilities $P(f)$ for functions f defined on all 128 inputs of a $n = 7$ Boolean system upon random sampling of parameters of an FCN with 10 layers and hidden width 40 (which is provably fully expressive for this system [1]), and tanh activation functions. The simplicity bias in $P(f)$ becomes weaker as the width σ_w of the Gaussian parameter prior $P_{\text{par}}(\theta)$ increases. By contrast, for ReLU activations, the bias in $P(f)$ barely changes with σ_w (See Figure S3(a) [5]). The effect of the decrease in simplicity bias on DNN generalisation performance is demonstrated in Fig. 4.1c for a DNN trained to zero error on a training set S of size $m = 64$ using advSGD (an SGD variant taken from [64]), and tested on the other 64 inputs $x_i \in T$. The generalisation error (the fraction of incorrect predictions on T) varies as a function of the complexity of the target function. Although all these DNNs exhibit simplicity bias, weaker forms of the bias correspond to significantly worse generalisation on the simpler targets (see also Supplementary Note 10 [5]). For very complex targets, both networks perform poorly. For reference, we also show an unbiased learner, where functions f are chosen uniformly at random with the proviso that they exactly fit the training set S. Not surprisingly, given the $2^{64} \approx 2 \times 10^{19}$ functions that can fit S, the performance of this unbiased learner is no better than random chance.

The scatter plots of Fig. 4.1 (d)–(f) depict a more fine-grained picture of the behaviour of the SGD-trained networks for three different target functions. For each target, 1000 independent initialisations of the SGD optimiser, with initial parameters sampled from the Gaussian parameter prior $P_{\text{par}}(\theta)$ at width σ_w, are used. The generalisation error and complexity of each function found when the

DNN first reaches zero training error are plotted. Since there are 2^{64} possible functions that give zero error on the training set S, it is not surprising that the DNN converges to many different functions upon different random initialisations. For the $\sigma_w = 1$ network (where $P(f)$ resembles that of ReLU networks), the most common function is typically simpler than the target. By contrast, the less biased network converges on functions that are typically more complex than the target. As the target itself becomes more complex, the relative difference between the two generalisation errors decreases, because the strong inductive bias towards simple functions of the first network becomes less useful. No free lunch theorems for supervised learning tell us that when averaged over all target functions, the three learners above will perform equally badly [82, 83] (see also Appendix A.1.3).

4.1.1 Priors over complexity

To understand why relatively modest changes in the inductive bias towards simplicity lead to such significant differences in generalisation performance, we need another important ingredient, namely, how the *number* of functions varies with complexity. Basic counting arguments imply that the number of strings of a fixed length that have complexity K scales exponentially as 2^K [28]. Therefore, the vast majority of functions picked at random will have high complexity. This exponential growth of the number of functions with complexity can be captured in a more coarse-grained prior, the probability $P(K_{LZ})$ that the DNN expresses a function of complexity K upon random sampling of parameters over a parameter initialisation function $P_{\text{par}}(\theta)$, which can also be written in terms of functions as $P(K') = \sum_{f \in \mathcal{H}_{K'}} P(f)$, the weighted sum over the set $\mathcal{H}_{K'}$ of all functions with complexity $\tilde{K}(f) = K_{LZ}f$. In Fig. 4.1 (g) $P(K_{LZ})$ is shown for uniform random sampling of functions for 10^8 samples using the LZ measure, and also for the theoretical ideal compressor with $P(K) = 2^{K_{LZ} - (K_{LZ})_{max} - 1}$ over all $2^{128} \approx 3 \times 10^{38}$ functions (see also Appendix E.3.2). In (h) we display $P(K_{LZ}$ for functions not sampled at random, but rather from the two networks. There is a dramatic difference between random sampling functions (as in (g)) and between the network with $\sigma_w = 1$, where $P(K_{LZ})$ is nearly flat.

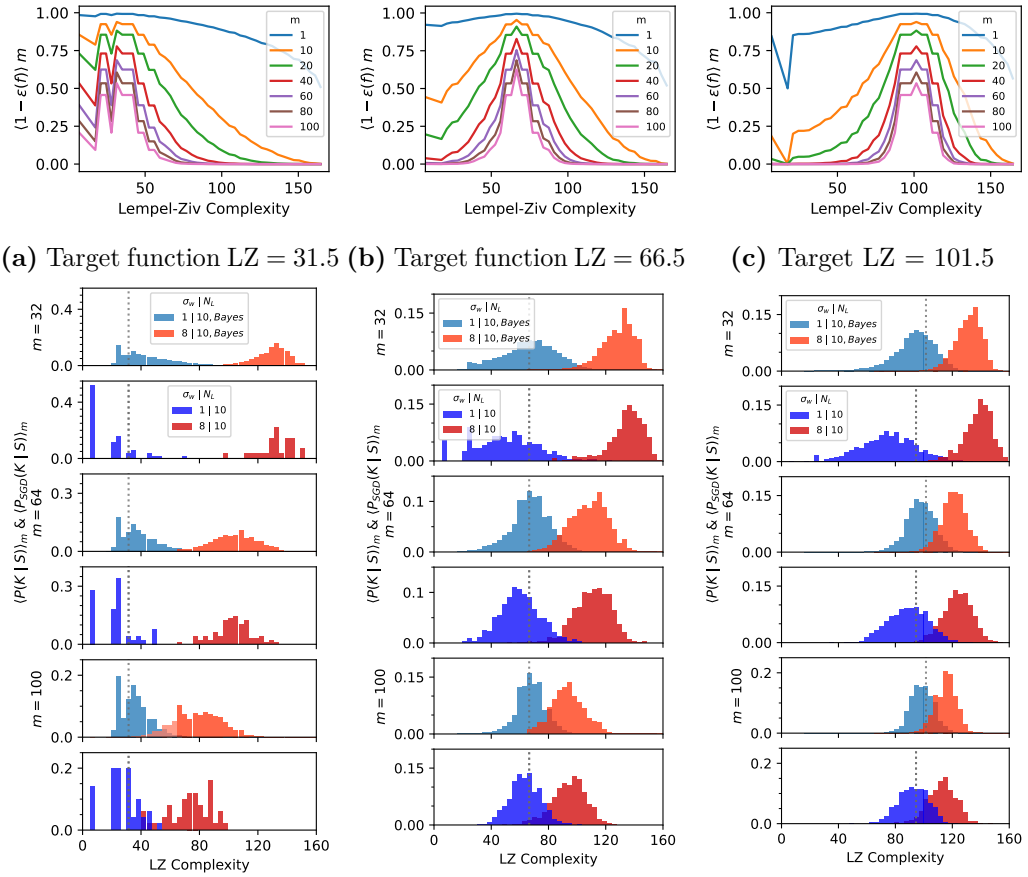
This behaviour follows from the interesting fact that the AIT coding theorem-like scaling [65, 64] of the prior over functions $P(f) \sim 2^{-\tilde{K}_{LZ}(f)}$ counters the $2^{K_{LZ}(f)}$ growth in the number of functions. See Appendix D for more on the relation between $P(K_{LZ}(f))$ and AIT, and relating these counting arguments to $K_{DNF}(f)$ from Chapter 3.

By contrast, even though, relative to the 38 or so orders of magnitude scale on which $P(f)$ varies, the more artefactual $\sigma_w = 8$ system has strong simplicity bias (we estimate that for the simplest functions, $P(f)$ is about 10^{25} times higher than the mean probability $\langle P(f) \rangle = 2^{-128} \approx 3 \times 10^{-39}$), this is not enough to counter the $2^{K_{LZ}}$ growth in the number of functions with complexity. Therefore, this DNN is exponentially more likely to throw up complex functions, an effect that SGD is unable to overcome.

More generally, the fact that the number of complex functions grows exponentially with complexity K_{LZ} lies at the heart of the classical explanation of why an insufficiently biased agent suffers from variance: It can too easily find many different functions that all fit the data. The marked differences in the generalisation performance between the two networks observed in Fig. 4.1 (c)–(f) can therefore be traced to differences in the inductive bias of the networks, as measured by the differences in their priors. We can formalise this using PAC-Bayes bounds – see Fig. E.8.

4.1.2 Artificially restricting model capacity

To further illustrate the effect of inductive bias, we create a K-learner (using LZ complexity, $K_{LZ}(f)$) that only allows functions with complexity $\leq K_M$ to be learned and discards all others. As can be seen in Fig. 4.1(i), the learners typically cannot reach zero training error on the training set if K_M is less than the target function complexity K_t. For $K_M \geq K_t$, zero training error can be reached, and not surprisingly, the lowest generalisation error occurs when $K_M = K_t$. As the upper limit K_M is increased, all three learning agents are more likely to make errors in predictions due to variance. The random learner has an error



(a) Target function LZ = 31.5 **(b)** Target function LZ = 66.5 **(c)** Target LZ = 101.5

(d) Target function LZ = 31.5 **(e)** Target function LZ = 66.5 **(f)** Target LZ = 101.5

Figure 4.2: How training data affects the posteriors: (a), (b), and (c) depict the mean likelihood $\langle (1-\epsilon_G(K_{LZ}))^m \rangle_5$ from Eq. (4.3), averaged over training sets, and over the 5 lowest error functions at each K. This term depends on data and is independent of the DNN architecture. With increasing m it peaks more sharply around the complexity of the target. In (d)–(f) we compare the posteriors over complexity, $\langle P_{\text{SGD}} K_{LZ}(f)|S) \rangle_m$, for SGD (darker blue and red) averaged over training sets of size m, to the prediction of $\langle P(K_{LZ}(f)|S) \rangle_m$ from Eq. (4.3) (lighter blue and orange), calculated by multiplying the Bayesian likelihood curves in Figs (a)–(c) by the prior $P(K_{LZ}(f))$ shown in Fig. 4.1(h). The light (Bayes) and dark (DNN) blue histograms are from the $\sigma_w = 1$ system, and the orange (Bayes) and red (DNN) histograms are from the $\sigma_w = 8$ system, which has less bias towards simple functions. The Bayesian decoupling approximation (Eq. (4.3)) captures the dominant trends in the behaviour of the SGD-trained networks as a function of data complexity and training set size. Quantitative measures of the similarity between the posteriors can be found in Figure S22 of [5].

that grows linearly with K_M. This behaviour can be understood with a classic Probably Approximately Correct (PAC) bound [84] where the generalization error (with confidence $0 \leq (1-\delta) \leq 1$) scales as $\epsilon_G \leq (\ln |\mathcal{H}_{\leq K_M}| - \ln \delta)/m$, where $|\mathcal{H}_{\leq K_M}|$ $K \leq K_M$ is the size of the hypothesis class of all functions with $K \leq K_M$;

the bound scales linearly in K_M, as the error does (see Appendices A.5 and E.1.13 for further discussion including the more sophisticated PAC-Bayes bound [85, 63].). The generalisation error for the $\sigma_w = 1$ DNN does not change much with K_M for $K_M > K_t$ because the strong inductive bias towards simple solutions means access to higher complexity solutions doesn't significantly change what the DNN converges on.

See also Figures S6-S11 in [5] for DNNs in the ordered regime with $\sigma_w \ll 1$, and for other optimisers, loss functions, and activation functions. These results broadly exhibit the same behaviour we describe here.

4.2 Calculating the Bayesian posterior and likelihood

To better understand the generalisation behaviour observed in Fig. 4.1 we apply Bayes' rule, $P(f|S) = P(S|f)P(f)/P(S)$ to calculate the Bayesian posterior $P(f|S)$ from the prior $P(f)$, the likelihood $P(S|f)$, and the marginal likelihood $P(S)$. Again, $K = K_{LZ}(f)$ is used. Since we condition on zero training error, the likelihood takes on a simple form. $P(S|f) = 1$ if $\forall x_i \in S, f(x_i) = y_i$, while $P(S|f) = 0$ otherwise. For a fixed training set, all the variation in $P(f|S)$ for $f \in U(S)$, the set of all functions compatible with S, comes from the prior $P(f)$ since $P(S)$ is constant. Therefore, in this Bayesian picture, the bias in the prior is translated over to the posterior.

The marginal likelihood also takes a relatively simple form for discrete functions, since $P(S) = \sum_f P(S|f)P(f) = \sum_{f \in U(S)} P(f)$. It is equivalent to the probability that the DNN obtains zero error on the training set S upon random sampling of parameters, and so can be interpreted as a measure of the inductive bias towards the data. The Marginal-likelihood PAC-Bayes bound [63] makes a direct link $P(S) \lesssim e^{-m\epsilon_G}$ to the generalization error ϵ_G which captures the intuition that, for a given m, a better inductive bias towards the data (larger $P(S)$) implies better performance (lower ϵ_G).

One can also define the posterior probability $P_{\mathrm{SGD}}(f|S)$, that a network trained with SGD (or another optimiser) on training set S, when initialised with $P_{\mathrm{par}}(\theta)$,

converges on function f. For simplicity, we take this probability at the epoch where the system first reaches zero training error. Note that in Fig. 4.1 (d)–(f) it is this SGD-based posterior that we plot in the histograms at the top and sides of the plots, with functions grouped either by complexity, which we will call $P_{\mathrm{SGD}}(K|S)$, or by generalization error ϵ_G, which we will call $P_{\mathrm{SGD}}(\epsilon_G|S)$.

DNNs are typically trained by some form of SGD, and not by randomly sampling over parameters, which is much less efficient. However, a recent study [2] which carefully compared the two posteriors has shown that to first order, $P_{\mathrm{B}}(f|S) \approx P_{\mathrm{SGD}}(f|S)$, for many different data sets and DNN architectures. See Figures S14-S15 in Mingard et al. [5] for an explicit demonstration of our $n = 7$ Boolean system. This evidence suggests that Bayesian posteriors calculated by random sampling of parameters, which are much simpler to analyse, can be used to understand the dominant behaviour of an SGD-trained DNN, even if, for example, hyperparameter tuning can lead to 2nd-order deviations between the two methods (see also Appendix B.4).

To test the predictive power of our Bayesian picture, we first define the function error $\epsilon(f)$ as the fraction of incorrect labels f produces on the full set of inputs. Next, we average Bayes' rule over all training sets S of size m:

$$\langle P(f|S)\rangle_m = P(f)\left\langle \frac{P(S|f)}{P(S)}\right\rangle_m \approx \frac{P(f)\,(1-\epsilon(f))^m}{\langle P(S)\rangle_m} \tag{4.2}$$

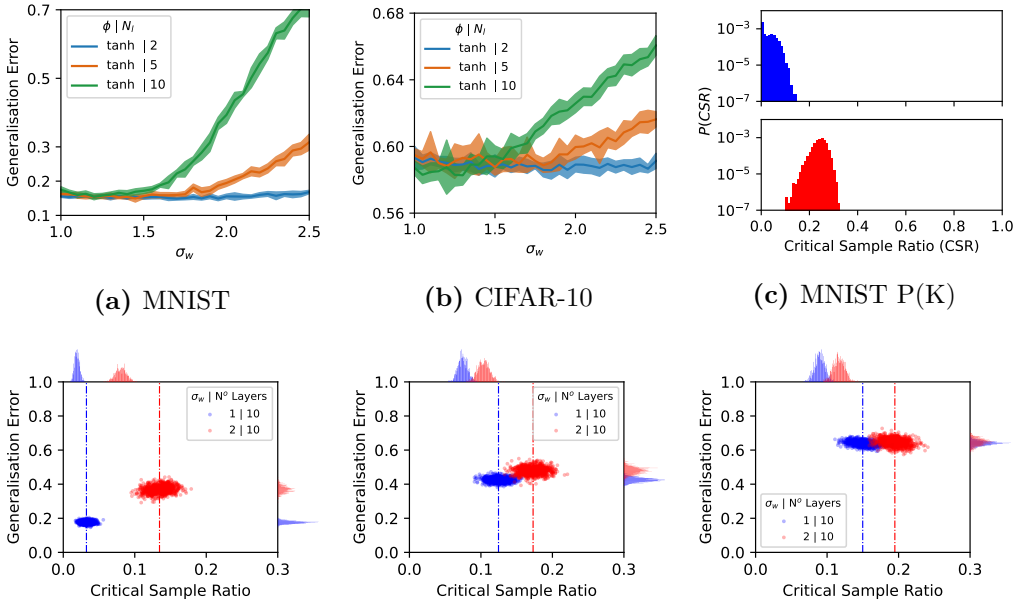
where the mean likelihood $\langle P(S|f)\rangle_m = (1-\epsilon(f))^m$ is the probability of a function f obtaining zero error on a training set of size m. In the second step, we approximate the average of the ratio with the ratio of the averages, which should be accurate if $P(S)$ is highly concentrated, as is expected if the training set is not too small.

Eq. (4.2) is hard to calculate, so we coarse-grain it by grouping together functions by their complexity:

$$\langle P(K|S)\rangle_m = \sum_{K(f)=K} \langle P(f|S)\rangle_m \propto P(K)\langle (1-\epsilon_G(K))^m\rangle_l, \tag{4.3}$$

and in the second step make a *decoupling approximation* where we average the likelihood term over a small number l of functions with complexity K with

lowest generalization error $\epsilon_G(K)$ since the smallest errors in the sum dominate exponentially since $(1-\epsilon_G) \approx e^{-\epsilon_G}$ for $|\epsilon_G| \ll 1$. We then multiply by $P(K)$, which takes into account the value of the prior and the multiplicity of functions at that K, and normalise $\sum_K P(K|S) = 1$. For a given target, we make the ansatz that this decoupling approximation provides an estimate that scales as the true (averaged) posterior.



(a) MNIST **(b)** CIFAR-10 **(c)** MNIST P(K)

(d) MNIST uncorrupted **(e)** MNIST 25% corruption **(f)** MNIST 50% corruption

Figure 4.3: MNIST and CIFAR-10 data. (a) MNIST generalization error for FCNs on a 1000 image training set versus σ_w for three depths. (b) CIFAR-10 generalisation error for FCNs trained on a 5000 image training set versus σ_w for three depths. The FCNs, made of multiple hidden layers of width 200, were trained with SGD with batch size 32 and lr=10^{-3} until 100% accuracy was first achieved on the training set. Error bars are one standard deviation. (c) Complexity prior $P(K_{CSR}(f))$, for 1000 MNIST images for randomly initialised networks of 10 layers and $\sigma_w = 1, 2$. Probabilities are estimated from a sample of 2×10^4 parameters. Figs (d), (e) and (f) are scatterplots of generalisation error versus $K_{CSR}(f)$ for 1000 networks trained to 100% accuracy on a training set of 1000 MNIST images and tested on 1000 different images. In (d), the training labels are uncorrupted, in (e) and (f), 25% and 50% of the training labels are corrupted, respectively. Note the qualitative similarity to the scatter plots in Fig 1 (d)-(f).

To test our approximations, we first plot, in Fig. 4.2 (a)–(c), the likelihood term in Eq. (4.3) for three different target functions using $K = K_{LZ}(f)$. To obtain these curves, we considered a large number of functions (including all functions with up to 5 errors w.r.t. the target, with further functions sampled). For each complexity,

we average this term over the $l = 5$ functions with smallest ϵ_G. Not surprisingly, functions close to the complexity of the target have the smallest error. These graphs help illustrate how the DNN interacts with data. As the training set size m increases, functions that are most likely to be found upon training to zero training error are increasingly concentrated close to the complexity of the target function.

To test the decoupling approximation from Eq. (4.3), we compare in Fig. 4.2 (d)-(f) the posterior $\langle P(K|S)\rangle_m$, calculated by multiplying the Bayesian likelihood curve from Fig. 4.2 (a)–(c) with the two Bayesian priors $P(K)$ from Fig. 4.1 (h) and (i), to the posteriors $\langle P_{\mathrm{SGD}}(K|S)\rangle_m$ calculated by advSGD [64] over a 1000 different parameter initialisations and training sets. It is remarkable to see how well the simple decoupling approximation performs across target functions and training set sizes. See Figures S13-14 in Mingard et al. [5] for a demonstration of the robustness of our approach, where we show that using $l = 1$ or $l = 50$ functions does not change the predictions significantly. This success suggests that our simple approach captures the essence of the interaction between the data (measured by the likelihood, which is independent of the learning algorithm) and the DNN architecture (which is measured by the prior and is independent of the data).

We have therefore separated out two of the three parts of the tripartite scheme, which leaves the training algorithm. In the figures above, our Bayesian approximation captures the dominant behaviour of an SGD-trained network. This correspondence is consistent with the results and arguments of [2]. See Fig. S15 in [5] for a similar set-up using MSE loss, where Bayesian posteriors can be exactly calculated using Gaussian processes (GPs). The direct Bayesian GP calculation closely matches SGD-based results for our much smaller network. Note that, in the spirit of model calculations, as called for in [86, 87], we mainly used a much smaller DNN. But their agreement with the GP-based posteriors, calculated for the infinite width limit, shows that at the scale of our Bayesian approach to the 1st-order generalisation question we are addressing here, the size of the DNN is not an important factor. The width of a DNN can, of course, be a factor for 2nd order generalisation questions.

4.2.1 Beyond the Boolean model: MNIST & CIFAR-10

Can the principles worked out for the Boolean system be observed in larger systems that are closer to the standard practice of DNNs? To this end, we show, in Fig. 4.3 (a) and (b), how the generalisation error for the popular image datasets MNIST and CIFAR-10 changes as a function of the initial parameter width σ_w and the number of layers N_l for a standard FCN, trained with SGD on cross-entropy loss with tanh activation functions. Larger σ_w and larger N_l push the system deeper into the chaotic regime [79, 80] and result in decreasing generalisation performance, similar to what we observe for the Boolean system for relatively simple targets. In Fig. 4.3, we plot the prior over complexity $P(K)$ for a complexity measure called the critical sample ratio (CSR), $K_{CSR}(f)$ [72], an estimate of the density of decision boundaries that should be appropriate for this problem. Again, increasing σ_w greatly increases the prior probability that the DNN produces more complex functions upon random sampling of parameters. Thus, the decrease in generalisation performance is consistent with the inductive bias of the network becoming less simplicity biased, and therefore less well aligned with structure in the data. Indeed, datasets such as MNIST and CIFAR-10 are thought to be relatively simple [88, 89].

These patterns are further illustrated in Fig. 4.3 (d)–(f), where we show scatterplots of generalisation error vs. CSR complexity for three target functions that vary in complexity (here obtained by corrupting labels). The qualitative behaviour is similar to that observed for the Boolean system in Fig. 4.1. The more simplicity-biased networks perform significantly better on the simpler targets, but the difference with the less simplicity-biased network decreases for more complex targets. While we are unable to directly calculate the likelihoods because these systems are too big, we argue that the strong similarities to our simpler model system suggest that the same basic principles of inductive bias are at work here.

4.3 Remarks

In order to generalise, high-capacity models need a clear inductive bias towards functions that perform well on the data being studied [90]. Here we show that DNNs (and also NNGPs and DNN-inspired kernels have a very specific kind of inbuilt Occam's razor (see Appendix A for background on Occam's razor(s)). This inductive bias can be quantified by an AIT coding theorem-like scaling of the prior as $P(f) \propto 2^{-K(f)}$, which can counteract the 2^K growth of the number of functions with complexity. If this intrinsic inductive bias is slightly weaker, say $P(f) \propto 2^{-\alpha K(f)}$ with $\alpha < 1$, then it becomes much harder to overcome the 2^K growth and the learner will likely suffer from strong variance problems. While we were not able to significantly increase the bias towards simplicity in DNNs, too strong a bias towards simplicity can mean bias (instead of variance) problems because complex functions become hard to find [91, 71].

Links to Solomonoff Induction An interesting direction to explore is the more formal arguments in AIT relating to the optimality of Solomonoff induction [28, 92, 93] (see also Appendix A.1.2). In particular, there may be fruitful links between simplicity bias, Solomonoff induction, and compression in deep learning, as recently discussed, for example, in the context of large language models [94, 95].

Links to kernels. Another promising direction for exploration is to connect our high-level theory here with the more detailed calculations of generalisation in kernels [96, 97, 98, 99, 100, 101] (see also Section 2.4 and Appendix B.2). These works can tell us *how* DNN-inspired kernels need to align with data, but not so easily *why* this would be so (the question of *why* neural networks have the inductive biases they do is, of course, addressed for one specific example in Chapter 3). An extra challenge is that our work here has been in the context of discrete functions and classification, whereas the work on kernels is typically for continuous regression scenarios. However, the classification setting allows us to study the inductive bias of a model over the entire space of input functions and simultaneously make

concrete generalisation bounds. We distinguished the first-order question of why high-capacity DNNs (and related models) generalise at all from the second-order question of how to further improve DNN performance. Firstly, the fact that DNNs are strongly biased towards simplicity gives a basis upon which to look for further inductive biases in other directions that may be orthogonal to simplicity. Our results are derived in a Bayesian setting, but do not preclude the fact that SGD optimisation itself can also introduce useful inductive biases.

Feature learning & Parameterisation In high-dimensional spaces, a Gaussian prior's weight is concentrated around its variance. For example, if we initialise a network in the chaotic regime with a weight standard deviation $\sigma_w = 1$ and then train it with aggressive weight decay, the optimisation process would push the weights back toward the ordered regime. At the end of training, the weights could have a norm closer to 1 rather than 2, which means the prior distribution P(f) for $\sigma_w = 2$ would be less appropriate for the final model than one for $\sigma_w = 1$. We observed this effect in Chapter 3, where applying aggressive weight decay strengthened the bias towards l-parity functions for small l beyond the initialisation prior – in other words, by moving the weights a long way from initialisation, the optimiser induced a bias not captured by $P(f)$. In this chapter, our experiments used fairly standard hyperparameters and standard parameterisation. The strong agreement between our PAC-Bayes bounds and generalisation error, as well as the predictive power of our priors over complexity for optimiser-trained neural networks, suggests that for FCNs on Boolean data, the optimiser primarily samples the solution space local to the initial scale. We will investigate the effects of optimiser hyperparameters in Chapters 6 and 7.

Finally our observations on inductive bias can be inverted to identify characteristics – such as limits on the complexity – of the data that DNNs can successfully learn. In particular, the remarkable success of DNNs on a broad range of scientific problems [102, 103, 104, 105] suggests that their inductive biases must recapitulate something deep about the structure of the natural world [73]. By

understanding why DNNs select specific solutions, we may, in turn, gain profound insights into the structure of nature itself.

5
Zipf's law

In this chapter, we show that the priors of high capacity learning agents, like neural networks and kernel methods, almost always exhibit Zipf's law. We argue that this condition is necessary for optimal learning.

Mingard, C., Ridout, S.*, Grabarczyk R., London, C., Dingle, K., Nemenman, Ilya and Louis, A A. Successful high capacity machine learning models exhibit a universal Zipf's law. Unpublished (to be submitted in late 2025).* [11]

In Chapter 4 we analysed the prior of continuous neural networks on the Boolean data, and found that $P(f) \lesssim 2^{-aK(f)}$ can be controllably weakened by tuning the initial weight variance, σ_w to move the network from the ordered regime ($a = 1$) into the chaotic regime (where $a < 1$). We observed Zipf's law in the prior. One might expect this from basic counting arguments: if there are $\sim 2^K$ functions with complexity K, and $P(f) \sim 2^{-K}$, then the rank $R(f) \sim 2^K$ and thus $P(f)R(f) \sim \mathcal{O}(1)$. In this chapter we study this in greater detail.

Generalising from experiences to new situations requires assumptions. If a natural or artificial learning agent generalises well, it is only because its assumptions match reality. For example, consider binary classification of $m = m_{\text{test}} + m_{\text{train}}$ *distinct* inputs. There are 2^m distinct classifier functions, denoted f, on this input space. If m_{train} of the inputs (with labels) are used as training data, then $2^{m_{\text{test}}}$ of these functions fit the training data perfectly. A sufficiently expressive learning

machine can produce *any* of these functions. Thus, in addition to the target function, which has zero error on both the training and test sets, it can express any of the other $2^{m_{\text{test}}} - 1$ functions with zero training error, most of which have poor generalisation (test) accuracy. Since these functions all have zero training error, only a prior bias can privilege the "correct" function over the others. Thus, in learning there is no free lunch: an agent or algorithm can only learn if it has an "inductive bias" which prefers certain functions over others, and only if this bias happens to align with true data-generating process [83, 106].

The classical solution to this problem is *capacity control* (see Appendix A): using a learning machine which is not fully expressive (i.e., cannot produce all 2^m classifications). In this context, "expressiveness" is measured through quantities such as the VC dimension or Rademacher complexity, and when the expressiveness is sufficiently small compared to the amount of training data, one can prove PAC ("probably approximately correct") bounds on the difference between the training and test errors.

In contrast, modern machine learning architectures often perform well even when they have more parameters than training examples. For example, in this regime, large neural networks can fit even random training data [107], suggesting that they can express *any* of the 2^m possible functions. When they generalise well, they must have a strong bias towards a small, low-test-error subset of the $2^{m_{\text{test}}}$ functions with zero training error.

In Bayesian methods, an inductive bias is directly encoded in a Bayesian prior (see Section 2.2). In most machine-learning methods, however, this bias is encoded *implicitly*, through some combination of the architecture of the learning machine, initialisation procedure, the minimization algorithm, and the distribution of the training data.

Here we consider expressive learning machines with parameters θ, that take an input x and output a binary classification $F(x;\theta)$. For example, $F(x;\theta)$ may be the output of a (thresholded) neural network (NN). Consider a fixed dataset $S = \{(x_i, y_i)\}_{i=1}^m$, with S_X and S_Y referring to the set of inputs and labels

respectively. There are 2^m possible classifier functions on S_X, each represented by an m-dimensional vector $f = (f_1, \ldots, f_m)$.

As in the previous two chapters, we define the "initialisation prior" $P(f)$ as the probability that f is expressed upon sampling a parameter initialisation distribution $P_{\text{par}}(\theta)$, i.e.,

$$P(f) = \int \mathrm{d}\theta \, P_{\text{par}}(\theta) \prod_{i=1}^{i=m} \mathbb{1}(F(x_i; \theta) = f_i), \tag{5.1}$$

where $\mathbb{1}$ is an indicator function, i.e., it is 1 if $F(x_i, \theta) = f_i$ and 0 otherwise.

$P(f)$ does not describe the training procedure, but contains information about the inductive bias produced by the data distribution, architecture, and initialisation procedure. It directly describes the inductive bias of a very simple learning algorithm: drawing new parameters θ from the initialisation distribution until a solution with zero training error is found. This is also equivalent to Bayesian inference with $P(f)$ as a prior and a "0–1" likelihood, $\mathbb{1}[f_i = y_i]$.

For NNs trained through gradient-based methods, we expect the initialisation prior to still encode information about the inductive bias because the parameters of wide NNs tend not to change much during training. Thus, of the many functions which fit the training data, NNs are likely to learn one in a region that is well sampled by the initialisation distribution. Indeed, this prior has been used to explain key aspects of NN generalisation performance [64, 2, 5].

Below, we show that, for a series of expressive machine-learning models that generalise well, the initialisation prior $P(f)$ has a universal power-law form $P(f) \propto 1/\operatorname{rank} f$ ("Zipf's law"), where $\operatorname{rank} f$ is the "rank" of f, i.e., 1 for the function which maximizes $P(f)$, 2 for the next most likely function, etc. We develop two explanations for this observation. Firstly, we show how Zipfian priors generically emerge in the kernel limit of NNs, through a low-dimensional latent variable which encodes the inductive bias [108, 109, 110]. Secondly, we show that inference with any flatter power-law prior guarantees poor generalisation to unseen data, and inference with any steeper power-law prior produces larger training errors than expected for a given likelihood. Thus, a Zipfian prior may be necessary (but not sufficient) for

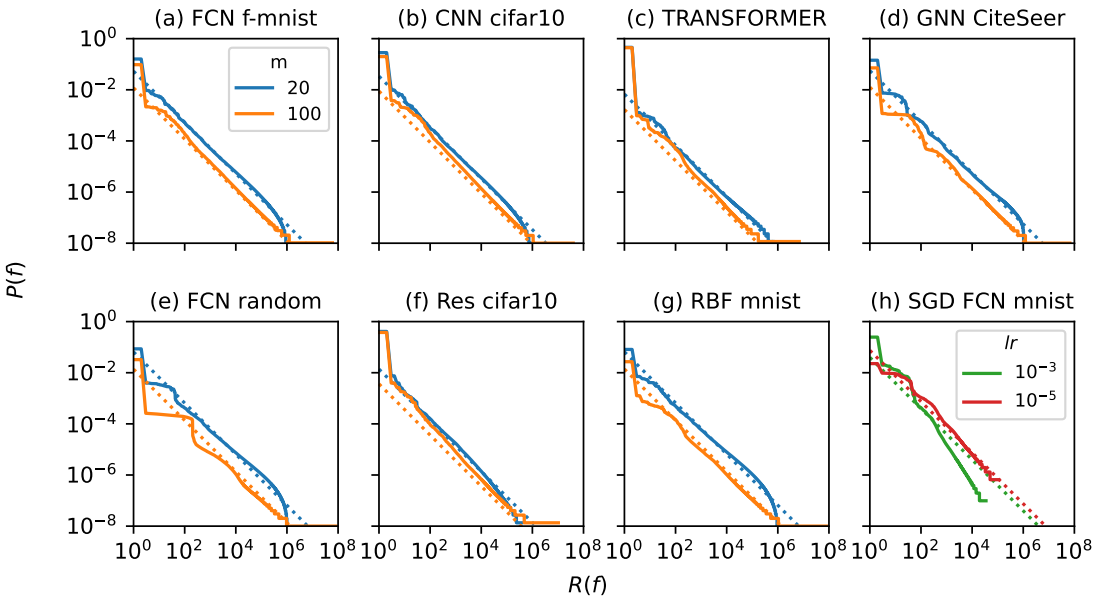


Figure 5.1: Priors for different learning systems, architectures, and datasets follow Zipf's law. (a–f) show the initialisation priors of neural networks, $P(f)$ (see Eq. (5.1)) vs. rank. (g) shows the prior of an RBF kernel, and (h) shows the effective prior induced by SGD, $P_{\text{SGD}}(f|\mathcal{S})$, for learning rates of 10^{-3} and 10^{-5} (see Section 5.4). Solid lines denote empirical data for 10^8 random samples for datasets of size $m = 20$ (blue) and $m = 100$ (orange). The $m = 20$ curve suggests three regions, (1) an initial blocky fluctuating region up to a rank of order m, (2) an extended region with a $1/\text{rank}\, f$ Zipf's law decay, and (3) a tail that falls off more rapidly at the very highest ranks. For $m = 100$, we only observe a tiny fraction of all functions, so regime (3) is not observed. The dashed lines are the parameter-free normalised $P(f) = C/\text{rank}\, f$ from Appendix F.1.2, which fit the empirical data remarkably well. See Appendix F.4 for more datasets and architectures, and Appendix F.3 for SGD priors with higher learning rates.

optimal learning. Supporting these results, we empirically show that models that are known to generalise poorly have priors that decay more slowly than Zipf's law.

5.1 A universal Zipf's law in the initialisation prior

In Fig. 5.1(a–g), we show $P(f)$ vs. rank f for a series of learning architectures and datasets. We consider the cases of $m = 20$ and $m = 100$ inputs, and, to keep the number of functions from becoming too large, binarize the labels. There are thus $2^{20} \approx 10^6$ and $2^{100} \approx 10^{30}$ possible functions, respectively. For each system, we took 10^8 samples. For architecture/dataset details, see Appendix F.1.1. Other

examples, with similar results, are in Fig. F.4.

We observe a striking universality in $P(f)$, for all architecture/data combinations. Except for the few most likely functions, whose probability is sensitive to details of the architecture and dataset (see Fig. F.4e), all cases show "Zipf's law" scaling, $P(f) \propto 1/\operatorname{rank} f$.

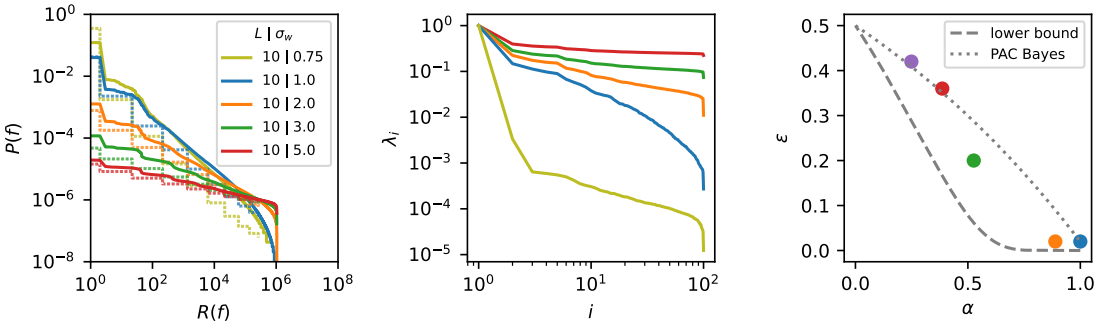
For $m = 20$, where we have enough samples to see all $2^{20} \approx 10^6$ functions, this scaling is only cut off at the edge of the distribution, where $\operatorname{rank} f \approx 10^6$. For $m = 100$, we cannot sample the vast majority of the functions. To find evidence that Zipf's law may persist over the unobservable ranks, we examine not just the *scaling* $P(f) = C/\operatorname{rank} f^{\alpha}$, but the normalization C. After excluding the (few) t most likely functions, we determine C theoretically by assuming Zipf's law holds for *all* ranks. This normalization fits the data well (dashed lines in Fig. 5.1), and is insensitive to t for $t \geq 2$ (Fig. F.5). Importantly, the theoretical C is very sensitive to α: e.g., for $m = 100$, taking $\alpha = 1.1$ would increase C sevenfold (Appendix F.12). This suggests that the Zipfian scaling continues beyond observed ranks.

The observed universality of the Zipfian scaling in the initialisation prior is striking. Nevertheless, different architectures differ: Fig. F.6 shows that the *rank ordering* of functions differs between an FCN, CNN and Resnet18 (on CIFAR-10 and MNIST). This ordering encodes the particular bias of each architecture, determining differences in performance. In other words, there's enough freedom of ordering within the (nearly universal) Zipf's law to account for differences between learning machines.

We can imagine two types of explanation for the universality of Zipf's law in $P(f)$. *Proximately*, a common mechanism related to their size and structure may produce Zipfian priors in varied NNs. *Ultimately*, perhaps non-Zipfian priors are inferior for learning, and thus architectures or hyperparameters that produce them never caught on.

5.2 Where does Zipf's law come from?

We begin with the proximate question: how, mechanistically, do learning machines produce Zipfian priors?



(a) (b) (c)

Figure 5.2: Priors that decay more slowly than Zipf's law lead to poor learning performance. (a) A 10-layer FCN of width 1024 with tahn activations is initialised in the chaotic regime, leading to a power law exponent α of the prior $P(f)$ vs. rank (solid lines) that shrinks towards 0 as the weight initialisation width σ_w increases. The dotted lines show an approximation using an equicorrelated kernel (Appendix F.8). (b) Emergence of subzipfian priors in this limit is associated with reduced decay in the eigenspectrum. (c) The test accuracy on 10000 datapoints from binarised MNIST (green dots) drops significantly with decreasing α, and lies between our approximate PAC–Bayes upper bound, and the lower-bound from Eq. (5.3). We assumed that $P(f) \propto \text{rank}\, f^{-\alpha}$, with the value of α fit to the data from panel a. To compute the PAC-Bayes (upper) bound, we also need to determine the probability of the true function f^*, or equivalently its rank f^*. As this is impossible to measure for $m = 10000$ inputs, we estimated it by assuming that the PAC-Bayes bound is tight at $\sigma_w = 1$ and that rank f^* is independent of σ_w. This is likely an *underestimate* of the upper bound. The resulting estimate is rank $f^* = 2^{278}$ (compared to the maximum rank 2^{10000}). Our lower bound does not depend on this estimate of rank f^*.

In particular, in the limit of infinite width, neural networks become equivalent to kernel machines which, despite their simplicity, encode inductive biases that allow them to generalise well [98, 96, 111]. Can we see how such simple models produce Zipf's law?

Past work has shown that Zipf's law emerges when high-dimensional observations are driven by a lower-dimensional "latent variable" whose distribution is broad [108, 109]. In particular, Zipf's law emerges, as the dimension $m \to \infty$ when $\delta \log P \sim m^2$ (when functions are sampled according to $P(f)$) [109]. This is because both faster and slower decays of $P(f)$ produce lower variances. Intuitively: If $P(f)$ is flatter, all functions have similar $\log P$. If $P(f)$ is steeper, functions have very different $\log P$, but the *sampled* functions are always the high P ones.

In turn, this $O(m^2)$ variance requires a very strong correlation between the m

elements of f, and most naturally emerges from a latent variable with $d \ll m$, because independently summing the effects of $O(m)$ variables would instead give $O(m)$ variance.

An infinite-width neural network's initialisation prior is equivalent to a Gaussian Process with a corresponding kernel K. The presence of Zipf's law in different NN architectures suggests that the mechanism producing it must be very general. Thus, we will study the priors of kernel machines, expecting our results to also apply qualitatively to finite-width NNs.

In the context of kernel regression, other work suggests that a "strong enough" inductive bias to enable generalisation comes from sufficiently fast decay of the kernel eigenvalues, and that generalisation then occurs for target functions which are well-aligned with this inductive bias [100, 112, 96, 98]. In this picture, at a given sample size m, a certain number of high-eigenvalue kernel eigenfunctions are "learnable" (if they are present in the target function), and the remainder are not, and are "benignly overfit" [113] to the training data.

In particular, kernels with eigenvalues that decay as a power law, $\Lambda_j \sim 1/j^\ell$ with $\ell > 1$, encode a sufficient inductive bias to allow for generalisation. In Appendix F.8 we study this model, under mild simplifying assumptions (we consider classification using a softmax rather than the sign of the output, and assume the unlearnable, low-eigenvalue eigenvectors are random).

To define a latent variable, we artificially split the eigenvectors of the kernel into a "learnable" set of the $m^a, a < 1$ with the largest eigenvalues, and the "unlearnable" remainder. A detailed calculation then shows that (1) the contributions of the "unlearnable" modes to $\log P$ are subdominant and can be ignored, and (2) the "learnable" modes then produce $O(n^2)$ variance in $\log P$. Thus, the projections of a function onto this learnable set of eigenvectors act as the latent variables which produce Zipf's law.

In the limit of infinite depth, the kernel approaches a particularly simple form—an equicorrelated kernel $K_{ij} = \delta_{ij} + (1 - \delta_{ij})\rho$, where all pairs of outputs have the same correlation ρ (Appendix F.8.6).

We show in Appendix F.8.3 that this limit maps directly onto the model of [108], which has one latent variable, and thus produces Zipf's law at large ranks as $m \to \infty$. The persistence of Zipf's law for small m or rank f depends on the variance of this latent variable, which is determined by ρ (see Fig. 5.2). We show in Appendix F.8.3 that $\rho \geq 1/2$ produces a sufficiently broad distribution of the latent variable to produce Zipf's law even down to small ranks, and show in Fig. F.11 that all architectures in Fig. 5.1 produce off-diagonal kernel elements $K_{ij} \gtrsim 0.5$.

Note that the prior of an equicorrelated kernel is a *bad* prior—it retains no information about the structure of the input data. This clearly demonstrates that, while Zipf's law is necessary for a good prior, it is not sufficient. The simplicity of this prior may be inferred from the fact that it produces a sequence of "steps", of size $\binom{N}{k}$, which approximate $P(f) \sim 1/\operatorname{rank} f$. A high-dimensional latent variable (i.e., "more latent variables") produces a smoother Zipfian prior (Fig. F.10).

Past work connects the initialisation prior to a bias for neural networks toward "simple" functions [5]. Although we have not made this interpretation precise for kernels, the mechanism we have described is consistent with it—in our mechanism, low values of the latent variable (as in the explanation for the equicorrelated kernel, above) correspond to function classes with many functions, while high values of the latent variable correspond to "simple" function classes. In fact, it is straightforward to show that Bayesian model selection among model classes with widely varying complexities also gives a Zipfian prior (Appendix F.9).

Thus, in the kernel limit, we provide an answer to the "proximate" question: neural networks produce Zipf's law because, although they have an enormous number of parameters, their output at initialisation is well-described by a much lower-dimensional latent variable.

5.3 Zipfian priors are necessary for good learning performance

Kernel machines that learn well naturally produce Zipfian priors. Is this a mere accident? Or is the Zipfian nature of the prior related to learning performance?

Since no exact calculation links $P(f)$ to the trained network for a general architecture and learning algorithm, we begin with a simpler problem. We study binary classification via Bayesian inference with power-law priors $P(f) = C(m)/\operatorname{rank} f^{\alpha}$. A prior with $\alpha < 1$ has a weaker inductive bias than the Zipfian $\alpha = 1$, and $\alpha > 1$ a stronger inductive bias. This prior may be measured over either the full input space, with cardinality m, or over some training set, with cardinality m_{train}. We will return to gradient-based learning briefly in Section 5.4.

A hint that a Zipfian prior is special comes from the *predictive information* $I_{\text{pred}}(m_{\text{train}})$, the mutual information between a sample of m_{train} elements and all future observations [114], for data drawn from $P(f) = C(m)/\operatorname{rank} f^{\alpha}$. A short calculation shows that $I_{\text{pred}}(m_{\text{train}})$ remains finite as $m_{\text{train}} \to \infty$ for any $\alpha \neq 1$ (Appendix F.11), less than a model with one real-valued parameter [114]! However, extending these calculations to a target distribution different from the model's prior is challenging, so we turn to other approaches to study the generalisation error.

5.3.1 Existing bounds fail to guarantee good generalisation for subzipfian priors

Learning theory usually seeks to bound the generalisation error of a model from above, guaranteeing good performance. We evaluate a PAC–Bayesian upper bound on the generalisation error for Bayesian inference [63], for a prior $P(f^{(\nu)}) \propto 1/\operatorname{rank} {f^{(\nu)}}^{\alpha}$. As $m_{\text{train}} \to \infty$, we find that this bound, instead of vanishing, is at least $(1-\alpha)\log 2$ for $\alpha < 1$ (Eq. (F.85)).

This means we cannot *prove* that learning is possible for a subzipfian prior, but does not prove that it is *impossible*. To do so, we derive *lower* bounds on the error.

5.3.2 A subzipfian prior guarantees nonzero generalisation error

Consider m possible inputs, of which we train on $m_{\text{train}} = qm$, with training error $\epsilon_S[\nu]m$. Define the test error $\epsilon_G[\nu]$ on the $(1-q)\,m$ unobserved inputs, i.e., the "off-training-set" error [83].

Consider Bayesian inference with a prior $P(f^{(\nu)}) \propto 1/\operatorname{rank} {f^{(\nu)}}^{\alpha}$ and likelihood

$$L(f^{(\nu)}) = e^{-\kappa q m \epsilon_S[\nu]}. \tag{5.2}$$

This is the likelihood derived by assuming that examples are mislabelled with probability $\epsilon_0(\kappa) = 1/(1+e^{\kappa})$. Taking $\kappa \to \infty$ gives the "hard" likelihood, analogous to training until zero training error is achieved.

α controls the strength of the inductive bias, while the rank ordering determines which functions are preferred. To derive a *lower* bound on $\epsilon_G[\nu]$, consider the best-case scenario for a given α, ordering functions by similarity to the target function. We derive equations for the test error ϵ_S^* of this "optimistic" ordering, for $m \to \infty$ (Appendix F.5), proving:

Theorem 2. *Bayesian inference with* $P_\nu \propto 1/\operatorname{rank} f^{(\nu)^\alpha}$ *and* $L_\nu = e^{-\kappa q m \epsilon_S[\nu]}$ *will give test error* $\epsilon_G[\nu] > \epsilon_G^* > 0$ *as* $m \to \infty$ *for* $\alpha < 1$.

For the hard likelihood, our equations simplify to

$$\frac{\log\left(\frac{1}{\epsilon_G^*(\kappa\to\infty)} - 1\right)}{\log\left(\frac{1}{(1-q)\epsilon_G^*(\kappa\to\infty)} - 1\right)} = \alpha, \quad \alpha < 1 \tag{5.3}$$

$$\epsilon_G^* = 0, \quad \alpha \geq 1. \tag{5.4}$$

ϵ_G^* is larger for smaller α and q (Fig. F.8).

For finite κ, solving our exact equations numerically shows that $\epsilon_G^*(\kappa)$ increases as κ is decreased (Fig. F.9).

These bounds generalise to priors which are not pure power laws, but are still flatter than $1/\operatorname{rank} f$ over a wide range of ranks (Appendix F.5).

5.3.3 A superzipfian prior underfits many target functions

A subzipfian prior lacks the inductive bias needed to generalise well. An unnecessarily strong inductive bias, however, may limit the number of functions that can be learned, leading to *underfitting*.

To see this, consider Bayesian inference with a soft likelihood (Eq. (5.2), $\kappa \neq \infty$). Since this likelihood comes from assuming data are mislabelled with rate $\epsilon_0(\kappa)$,

one expects $\epsilon_S = \epsilon_0(\kappa)$ if the prior contains the target function. If $\alpha = 0$, it is straightforward to show that $\epsilon_S = \epsilon_0(\kappa)$ for any target.

superzipfian priors, however, can give $\epsilon_S > \epsilon_0(\kappa)$. As a toy example, we considered a prior with functions in *random* order. In this case, a replica calculation gives $\epsilon_S = \epsilon_0(\kappa)$ for $\alpha \leq 1$ and $\epsilon_S = \epsilon_0(\kappa/\alpha) > \epsilon_0(\kappa)$ for $\alpha > 1$ (Appendix F.6.3).

We then showed that, for *any* ordering of functions in the prior, there are exponentially many functions for which, if κ is large enough, $\epsilon_S(\kappa) > \epsilon_0(\kappa)$.

Assume the prior is $P(f^{(\nu)}) = 1/\operatorname{rank} f^{(\nu)^\alpha}$ over the training set, and define $F(\epsilon')$ as the set of functions which differ from $f^{(1)}$ on $m_{\text{train}}\epsilon'$ training examples. We then show (Appendix F.6.2):

Theorem 3. *Let $\epsilon_{\min}(\alpha, \kappa, \epsilon')$ be defined as the solution to $S_2(\epsilon_{\min}) = S_2(\epsilon') - \frac{\kappa}{\alpha}\epsilon'$. Then for any $\epsilon < \epsilon_{\min}(\alpha, \kappa, \epsilon')$, as $m_{\text{train}} \to \infty$, all but an exponentially small fraction of the possible target functions in $F(\epsilon')$ will have training error at least ϵ.*

For any $\alpha > 1$, for sufficiently large κ, there is a range $\epsilon' \in (\epsilon'_1(\alpha, \kappa), \epsilon'_2(\alpha, \kappa))$ where $\epsilon_{\min}(\alpha, \kappa, \epsilon') > \epsilon_0(\kappa)$. This range grows wider as α or κ is increased.

Together, our results suggest that Zipfian priors are optimal for learning, thus providing an "ultimate" explanation for their universality.

5.3.4 The Solomonoff–Levin prior is asymptotically Zipfian

One may ask whether this "ultimate" picture—that Zipf scaling marks a critical point for learning—is tied only to the particular Bayesian classification setup above, or whether it already appears in the canonical formalisation of Occam's razor. We now show the latter: under standard algorithmic information theory (AIT), the Solomonoff–Levin universal semimeasure on finite strings is *asymptotically Zipfian* when strings are ranked by probability. The proof is a precise counting argument for the heuristic from the opening of this chapter, that a prior decaying as 2^{-K} against a multiplicity growing as $\sim 2^K$ forces $P \cdot \text{rank}$ to grow only polynomially in K; it is the rigorous counterpart of the coarse identity "$\text{rank}\, f \sim 2^{K(f)}$ with $P(f) \sim 2^{-K(f)}$" used for Boolean functions in Chapter 4. This complements

Section 5.2: there we gave kernel and latent-variable mechanisms for Zipf in concrete models; here the universal semimeasure provides an algorithmic-information *baseline* at the same asymptotic scaling.

We work with finite binary strings $x \in \{0,1\}^*$, prefix-free Kolmogorov complexity $K(x) = K_U(x)$ for a fixed prefix-universal machine U, and the Solomonoff–Levin semimeasure $P_U(x)$ from Section 2.3 (see also Appendix A.1.2). Levin's coding theorem [26, 28] gives

$$2^{-K(x)} \le P_U(x) \le 2^{-K(x)+d} \tag{5.5}$$

for a constant $d = d(U)$ independent of x (our Eq. (A.4)).

Define the *complexity counting function*

$$N(k) \;:=\; \Big|\{x \in \{0,1\}^* : K(x) \le k\}\Big|. \tag{5.6}$$

An upper bound on $N(k)$ follows from Kraft's inequality. There are exactly 2^j binary programs of length j, hence at most $\sum_{j=0}^{k} 2^j = 2^{k+1} - 1$ halting programs of length at most k, and each string x counted by $N(k)$ is the output of at least one such program. Distinct strings require distinct halting programs, so

$$N(k) \le 2^{k+1} - 1. \tag{5.7}$$

For a lower bound, fix any $n \ge 1$. A universal machine U can implement a fixed, always-halting routine that (i) reads a self-delimiting prefix encoding the integer n, then (ii) copies the next n input bits to the output. Such a self-delimiting code for n exists with length $K(n) \le \log_2 n + O(\log\log n)$ [28], so every length-n string x admits a prefix program of length at most $n + \log_2 n + O(\log\log n)$. Hence $K(x) \le n + \log_2 n + c$ for a constant c depending only on U. Choosing $n = k - \lfloor b \log_2 k \rfloor$ for a sufficiently large integer b yields $K(x) \le k$ for all 2^n strings x of that length, once k is large enough. Therefore

$$N(k) \;\ge\; 2^{\,k - b\log_2 k} \;=\; 2^{\,k - O(\log k)} \tag{5.8}$$

for all sufficiently large k. Combining Eqs. (5.7) and (5.8),

$$\log_2 N(k) = k + O(\log k) \qquad (k \to \infty). \tag{5.9}$$

Now order all finite strings by non-increasing P_U, breaking ties arbitrarily, and let $R_U(x) \in \mathbb{N}$ be the resulting rank (so $R_U = 1$ for the most probable string under P_U). Let $c := \lceil d \rceil + 1$, which depends only on U. If $K(x) = k$ and $K(y) \leq k - c$, then $P_U(y) \geq 2^{-K(y)} \geq 2^{-k+c}$ by the lower bound in Eq. (5.5), whereas $P_U(x) \leq 2^{-k+d}$. Because $c > d$, we have $2^{-k+c} > 2^{-k+d}$, so every such y strictly precedes x in the P_U-ordering. There are $N(k-c)$ strings of complexity at most $k - c$, hence

$$R_U(x) > N(k-c) \qquad \text{for } k \geq c. \tag{5.10}$$

For an upper bound, note that if $P_U(y) > P_U(x)$ then $P_U(y) > 2^{-k}$ because $P_U(x) \geq 2^{-k}$ by Eq. (5.5). Using the upper branch of Eq. (5.5), $P_U(y) \leq 2^{-K(y)+d}$, so any such y must satisfy $2^{-K(y)+d} > 2^{-k}$, i.e. $K(y) < k + d$. Hence there are at most $N(k + d - 1)$ predecessors, and

$$R_U(x) \leq N(k + d - 1). \tag{5.11}$$

Combining Eqs. (5.10) and (5.11) with Eq. (5.9),

$$\log_2 R_U(x) = K(x) + O\Big(\log K(x)\Big). \tag{5.12}$$

Substituting into the upper branch of Eq. (5.5) gives

$$P_U(x) = 2^{-K(x)+O(1)} = R_U(x)^{-1}\, 2^{O(\log K(x))} = R_U(x)^{-1}\, \mathrm{poly}\Big(K(x)\Big), \tag{5.13}$$

where the implicit degree of the polynomial depends only on U (through the $O(\log k)$ remainder in Eq. (5.9) and the constant d in Eq. (5.5)). Equivalently, $\log P_U(x) + \log R_U(x) = O(\log K(x)) = O(\log \log R_U(x))$: on a log–log plot of P_U versus rank, the slope tends to -1 as $K(x) \to \infty$. We will refer to this as *asymptotic Zipfian* behaviour of the universal prior.

In summary, the ideal algorithmic prior sits at the same asymptotic Zipf scaling as the critical exponent $\alpha = 1$ identified in our learning-theoretic results above, up to slowly varying multiplicative corrections. It does not by itself explain empirical initialisation priors of finite neural networks; for that, the kernel and latent-variable mechanisms of Section 5.2 remain the appropriate *proximate* account.

5.4 "Effective priors" for SGD are often Zipfian

DNNs are typically trained using stochastic gradient descent (SGD). The interaction of the initialisation prior with the stochasticity of the training means that, even if the training proceeds to (near) zero error, the result is stochastic, and the learned function is sampled from an induced distribution $P_{\mathrm{SGD}}(f|S)$. For moderate m, [2] shows that $P_{\mathrm{SGD}}(f|S)$ is similar to the Bayesian posterior of an appropriate Gaussian process. In Appendix F.3 we construct an *effective SGD prior*: a prior $P_{\mathrm{SGD}}(f)$ that, when used as a Bayesian prior with the 0–1 likelihood, generates a posterior that approximates $P_{\mathrm{SGD}}(f|S)$. This prior depends on the hyperparameters of SGD and the choice of the loss function. In Fig. 5.1(h), we show that, for an FCN architecture on the MNIST dataset with a learning rate of 10^{-5}, this SGD prior is also Zipfian. In Fig. F.3e we show that the ordering of functions for $P_{\mathrm{SGD}}(f|\mathcal{S})f$ is very similar to that in $P(f)$, supporting our earlier arguments that $P(f)$ is useful for studying even *trained* neural networks. Increasing the learning rate can make the approximate SGD prior steeper than Zipfian (see below, and Appendix F.3 for a discussion).

5.5 Non-Zipfian learning machines exist and show inferior performance

For bounded activation functions, DNNs have a well-defined transition between an ordered and a chaotic regime [79, 80]. For a Gaussian $P_{\mathrm{par}}(\theta)$ with standard deviation σ_w, chaotic behaviour emerges for large σ_w and large depth L. In Fig. 5.2(A) we show, for an FCN on MNIST, that $P(f)$ is subzipfian, with power $\alpha < 1$ decreasing as σ_w increases.

To understand this intuitively, consider the kernel limit discussed above. As $L \to \infty$, the kernel approaches the equicorrelated kernel, $K_{ij} = \delta_{ij} + (1 - \delta_{ij})\rho$. Further, as $\sigma_w \to \infty$, $\rho \to 0$, giving a uniform $P(f)$. (Appendix F.8.6). Fig. 5.2(B) shows that, indeed, smaller α is associated with smaller off-diagonal kernel elements, and the dashed lines in Fig. 5.2(A) show that an approximation using the equicorrelated kernel describes the observed subzipfian powers well.

In Fig. 5.2(C), we train this neural network. As σ_w is varied and α decreases, the generalisation error grows, as expected. Estimating the PAC–Bayes bound requires knowing the rank of the true function in the prior, rank f^*, which cannot be measured directly for large m. We estimate rank f^* by assuming the bound is saturated at $\sigma_w = 1$ and that rank f^* is independent of σ_w. We then find that the observed generalisation errors lie between our upper and lower bounds.

We have also observed subzipfian priors in perceptrons on boolean inputs as for this specific architecture-dataset combination, correlations between inputs are smaller than required (ρ is too small, see Fig. F.7). For power laws truncated at small rank, our lower bound on generalisation error becomes small (Appendix F.5), but the model will instead underfit.

We have not found neural networks with superzipfian initialisation priors. However, we have found that for large learning rates the effective prior produced by SGD can become superzipfian (Fig. F.3). Further, it is possible to design a Bayesian model selection algorithm with a superzipfian prior, by constructing a prior over model classes which is explicitly biased against complex models, beyond the usual "Occam factors" (Fig. F.12).

5.6 Remarks

In this chapter, we empirically demonstrated that Zipf's law appears in the prior of many well-initialised neural networks. We then established two main results. First, any kernel with a sufficiently rapidly decaying eigenspectrum (a condition satisfied by many NNGPs) will have a Zipfian prior, i.e. $P(f) \propto 1/R(f)$. This complements Cui et al. [100], who prove a similar result in the context of kernel methods learning continuous functions. Second, such a prior is required for optimal learning in the following setup: we considered a Bayesian learner with exponential likelihood in training error, and showed that errors due to variance occur for subzipfian priors, while errors due to bias occur for superzipfian priors in the large-data limit (except in very particular circumstances). This second result applies to all machine learning agents: we would expect successful, overparameterised learning agents to all have

Zipfian priors. We then attempted to extend these results to loss-based likelihoods (see Appendix F.2) and stochastic optimiser-trained neural networks using effective priors, and argued that in the large data limit, our main result should still apply.

Limitations and Future Work Our theoretical arguments connect a Zipfian initialisation prior to successful learning. Importantly, we proved only that the priors of kernel methods should be Zipfian, and only showed empirically that finite-width neural networks have Zipfian priors. Future work could look to bridge this gap.

Properly interpreting the effective prior is important to fully understand these results. Its behaviour broadly lines up with our intuitions at low learning rates, where we approximate Bayesian sampling [2] (c.f. Part III) and thus the effective prior should be close to the initialisation prior. At larger learning rates, the effective prior becomes superzipfian, something we also observed for GP training with sharp likelihoods. Understanding exactly why this happens, and explaining this behaviour in the large data limit, is an important next step.

Key open questions include whether Zipfian priors can be deliberately engineered during training, and whether Zipf-based diagnostics can reliably detect or predict underfitting and overfitting in practice. Extending the present analysis beyond supervised learning to unsupervised, generative, or reinforcement settings may clarify whether Zipf's law is a universal property of all successful learning systems. Finally, exploring deeper connections to formal complexity measures such as Kolmogorov complexity or Minimum Description Length could place Zipfian priors on a more rigorous algorithmic foundation.

Part III

Feature learning & Optimisation

The previous part of this thesis explained generalisation through the lens of inductive biases at initialisation. While this perspective aligns naturally with Bayesian inference, its connection to networks trained with standard optimisers is less direct. This part bridges that gap by analysing the training dynamics themselves. In Chapter 6, we first show that for wide networks on simple tasks, SGD's dynamics closely approximate Bayesian inference, and we explore the conditions under which this approximation breaks down and we begin to learn different features than those at initialisation. We then introduce a novel framework for quantifying feature learning, using it to systematically analyse how different optimisers and architectural choices impact the learned representations. Finally, in Chapter 7, we move from analysing existing optimisers to deriving a new one from first principles. We propose a theoretical framework that extends mirror descent to explicitly incorporate neural architecture, a feature neglected by current methods. This yields Automatic Gradient Descent (AGD), a first-order, hyperparameter-free optimiser that successfully trains both fully-connected and convolutional networks at ImageNet scale, and acting as a rigorous foundation for new architecture-aware algorithms.

6
Feature Learning

In Chapter 3, we introduced a toy model: a discrete network (DFCN) on Boolean data. We were able to characterise its inductive prior $P(f)$ as a function of DNF complexity, $K_{DNF}(f)$, showing that complex functions have low $P(f)$.

We trained the DFCN with a Bayesian learning algorithm. Without a weight decay parameter, our posterior is (very close to) proportional to the prior – generally, the bias in $P(f)$ towards simple functions is enough for good generalisation. Increasing weight decay moves the solutions towards weights with low norm – where $P(f)$ is not concentrated. This encourages feature learning and, for simple functions like l-parity, significantly improves generalisation. In this case, $P(f)$ does not capture the entire inductive bias of this system – we can do better by feature learning.

In Chapters 4 and 5 we moved away from our easy-to-work-with toy model and towards continuous neural networks. We observed empirically that neural networks have an in-built bias towards simple functions (unlike the DFCN, we cannot prove this). We can explain how this inductive prior induces a bias towards simple functions in the posterior, but to do so rigorously, the learning algorithm has to be Bayesian. Once we introduce SGD, this rigour is no longer possible, and for $P(f)$ to be a good predictor of generalisation, we are reduced to requiring the following statements to hold "if SGD closely approximates a Bayesian learning algorithm with prior $P(f)$". Of course, we would like to make this precise.

The goal of this chapter is to investigate the effects of optimisers, asking (1) how do they fit into the neat simplicity-bias picture from Part II? and (2) when their solutions cannot be explained with linear models, what do they learn?

6.1 Is SGD a Bayesian Sampler? Well, almost

Note: the contents of this section were also submitted as a master's thesis, and are not to be taken as an original contribution to this thesis (and are instead included only for context).

Mingard, C., Valle-Pérez, G., Skalse, J. and Louis, A.A., 2021. Is SGD a Bayesian sampler? Well, almost. Journal of Machine Learning Research, 22(79), pp.1-64. [2]

Recall (from Section 2.7) that at initialisation a width-$\to \infty$ network with *Standard Parameterisation* (SP) induces a Gaussian–process prior over functions (the NNGP) with kernel K_{NNGP} (see Eq. (2.23)).

In contrast, in the *NTK parameterisation*—and, equivalently, when all layers are trained with infinitesimal step size while width $\to \infty$ so that the tangent kernel stays constant—the training dynamics linearise around initialization and mean–squared–error training is equivalent to kernel regression with the neural tangent kernel K_{NTK}.

Under SP, if one freezes the hidden layers and trains only the last layer (or uses an optimiser that effectively keeps features fixed), MSE training is equivalent to kernel regression with K_{NNGP}; the resulting predictor coincides with the GP posterior mean under the NNGP prior (this is not sampling). Alternatively, a fully Bayesian treatment with the NNGP prior yields a posterior over functions, which we denote $P_{\mathrm{B}}(f|\mathcal{S})$ (one may conceptually "sample from the prior and condition on the data" to obtain draws from this posterior).

In this section, we investigate how similar this Bayesian NNGP posterior $P_{\mathrm{B}}(f|\mathcal{S})$ is to the empirical distribution over test-set labellings induced by wide neural networks trained with SGD at a finite learning rate, $P_{\mathrm{SGD}}(f|\mathcal{S})$, where $P_{\mathrm{SGD}}(f|\mathcal{S})$

is defined by random initialisation and the optimisation procedure, conditioned on reaching (near-)zero training error.

Consider a neural network, parameterised by θ. We use the same notation as in Section 2.1: a training set $S = \{x_i, y_i\}_{i=1}^m$ and a test set $E = \{x'_i, y'_i\}_{i=1}^p$. We consider binary datasets for our experiments (multiclass datasets like CIFAR-10 and MNIST are binarised in our experiments). The neural networks express continuous functions on the data $g(x;\theta)$, where their thresholded functions $f(x;\theta) = \mathbb{1}[g(x;\theta) > 0]$. Note that f_S and g_S are defined on the training data only.

Sampling from the optimiser posterior For a given optimiser OPT (SGD or one of its variants), we sample initial parameters θ_{init}, from an i.i.d. truncated Gaussian distribution $P_{\text{par}}(\theta)$, and train with the optimiser until the first epoch where the network has 100% training classification accuracy.[1]. We then compute the function f found by evaluating the network on the inputs in E, as described before. As the dataset is binary, this sample is a vector of 1s and 0s of length p.

Note: this posterior $P_{\text{OPT}}(f|\mathcal{S})$ is not a Bayesian posterior; it is an empirical distribution induced by random initialisation, training dynamics and a stopping rule, conditioned on interpolating the training set.

Sampling from the Bayesian posterior We use an NNGP with square likelihood (as described in Section 2.7.1). See Appendix A.2.1 of [2] for more details on this particular implementation. As for the sample from $P_{\text{SGD}}(f|\mathcal{S})$ this sample is a vector of 1s and 0s of length p (we only keep the function on the test data as we require 100% accuracy on the training set).

Approximating the posterior The main innovation here is to discretise the function space by evaluating the classification on a finite test set. If we sample from both the NN and NNGP posterior as described above 10^5 times, we build up a good estimate of both $P_{\text{B}}(f|\mathcal{S})$ and $P_{\text{SGD}}(f|\mathcal{S})$ – provided both are biased

[1] If SGD fails to achieve 100% accuracy on S in a maximum number of iterations, we discard the run.

such that the same classification predictions show up many times. For an FCN on MNIST, with 10000 training datapoints and 100 test datapoints, this is the case: Fig. 6.1(a) shows that we obtain one particular function with 1 error occupies about half $P_{\mathrm{B}}(f|\mathcal{S})$ and $P_{\mathrm{SGD}}(f|\mathcal{S})$.

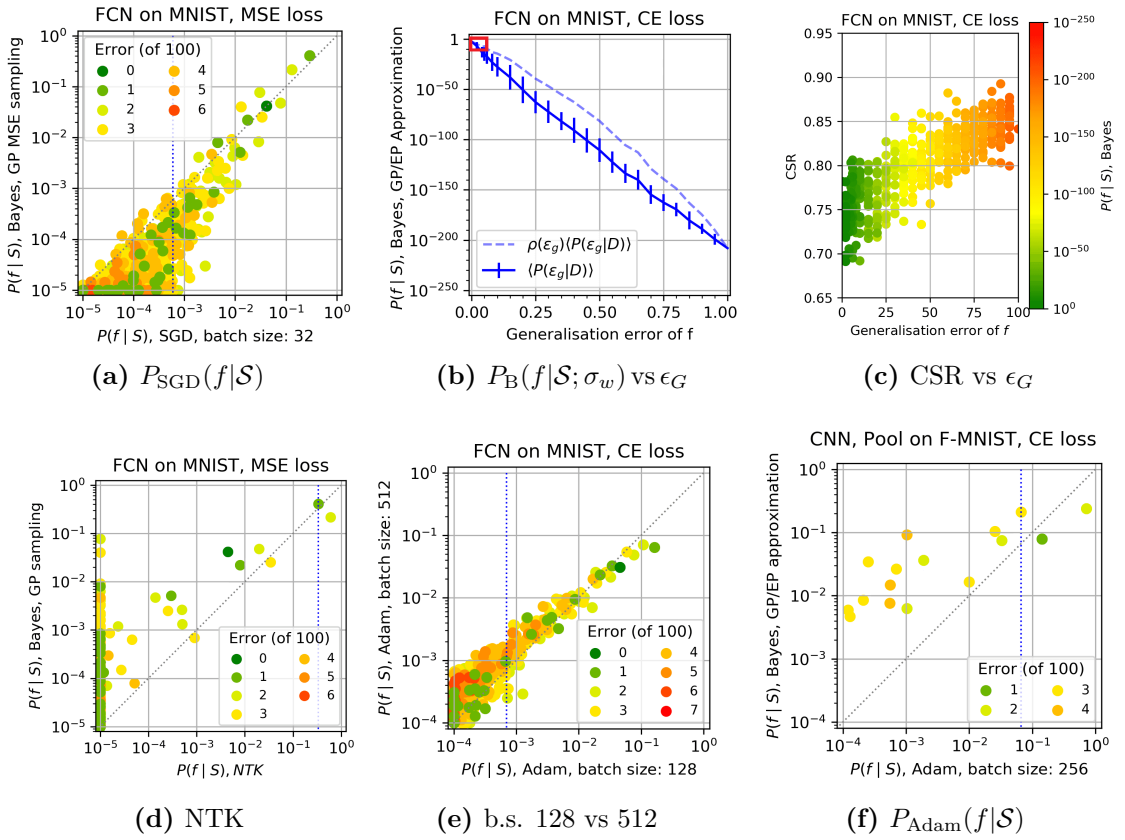
Summary of results Fig. 6.1 shows the main results from [2] with (a)-(e) showing a 3-layer, width-1024 FCN on MNIST with ReLU activations, and (f) a 2-layer CNN with pooling.

(a) shows $P_{\mathrm{SGD}}(f|\mathcal{S})$ is strongly correlated with $P_{\mathrm{B}}(f|\mathcal{S})$. The generalisation errors are similar at 1.88% and 1.61%, but clearly the two distributions could not be equal: at lower $P_{\mathrm{SGD}}(f|\mathcal{S})$ all the functions lie below the y=x line. [2] explored this further, and in Figures 17 and 20(e.f) showed that while the posterior distributions for any two optimisers remained correlated, the hyperparameters significantly affected their correlation.

(b) shows $P_{\mathrm{B}}(f|\mathcal{S})$ as a function of generalisation error. 20 functions were sampled uniformly per generalisation error, and we used the EP approximation to approximate $P_{\mathrm{B}}(f|\mathcal{S})$ for each. While this method does not give us a complete picture of the posterior, it backs up the observation in (a) that functions with larger errors have a lower $P_{\mathrm{B}}(f|\mathcal{S})$.

(c) uses a neural network-based complexity measure, Critical Sample Ratio, to link low $P_{\mathrm{B}}(f|\mathcal{S})$ and high error to larger complexity.

(d) shows that $P_{\mathrm{SGD}}(f|\mathcal{S})$ is strongly correlated with $P_{\mathrm{NTK}}(f|\mathcal{S})$, but less strongly to $P_{\mathrm{NTK}}(f|\mathcal{S})$ than $P_{\mathrm{B}}(f|\mathcal{S})$. The generalisation error for NTK is 1.69%, but the finer-grained analysis clearly shows a significant difference in the posteriors – there are two low-error functions with probabilities $P_{\mathrm{SGD}}(f|\mathcal{S}) = O(5\%)$ but $P_{\mathrm{NTK}}(f|\mathcal{S}) \lesssim 10^{-5}$. We explain the differences between the NNGP limit and NTK limit in Section 2.7.2.



(a) $P_{\mathrm{SGD}}(f|\mathcal{S})$ **(b)** $P_{\mathrm{B}}(f|\mathcal{S};\sigma_w)$ vs ϵ_G **(c)** CSR vs ϵ_G

(d) NTK **(e)** b.s. 128 vs 512 **(f)** $P_{\mathrm{Adam}}(f|\mathcal{S})$

Figure 6.1: Comparing the Bayesian prediction $\mathbf{P_B(f|S)}$ to $\mathbf{P_{OPT}(f|S)}$ for SGD and Adagrad, for an FCN on MNIST We use training/test set size of 10,000/100; For (a,e,f), the vertical dotted blue lines are drawn at the highest value of $P_{\mathrm{OPT}}(f|\mathcal{S})$ such that the sum of $P_{\mathrm{OPT}}(f|\mathcal{S})$ for all functions above the line is $> 90\%$ (90% probability boundary); dashed grey line denotes $P_{\mathrm{B}}(f|\mathcal{S};\sigma_w) = P_{\mathrm{OPT}}(f|\mathcal{S})$. See the main text for a full description of each subfigure.

(e) This experiment compares Adam with batch size 128 (test error 2.20%) to 256 (test error 2.67%). The respective posteriors $P_{\mathrm{Adam}}(f|\mathcal{S})$ are still well correlated, but clearly not exactly equal – the distribution is tilted from the y=x line, meaning the mode function is found more often by the smaller batch size. Figures 17 and 20 in [2] show greater deviation from $y = x$ for other optimiser pairs.

(f) Finally, we show $P_{\mathrm{B}}(f|\mathcal{S})$ vs $P_{\mathrm{Adam}}(f|\mathcal{S})$ for a CNN on Fashion MNIST. While there is still a strong correlation between $P_{\mathrm{B}}(f|\mathcal{S})$ and $P_{\mathrm{Adam}}(f|\mathcal{S})$, the deviation from y=x is significant, with the optimiser being even more biased towards functions with high $P_{\mathrm{B}}(f|\mathcal{S})$.

6.1.1 Remarks

For the wide networks studied in this section, the optimiser posteriors exhibit a strong correlation with the Bayesian posterior of the Neural Network Gaussian Process (NNGP). For this to be the case, weights in all but the last layer must remain close to initialisation. This is expected for wide neural networks with Standard Parameterisation (see Section 2.6). For these neural networks, the inductive bias induced by the prior over parameters is sufficient to understand generalisation.

6.2 Is SGD a Bayesian sampler? Depends on the neural network

> This section includes experiments not submitted as part of the master's thesis, follow on from [2] and are unpublished.

Experiments in Section 6.1 (and more broadly [2]) showed that for small-scale neural networks, the posterior of the optimiser can be well described by

$$\log P_{\text{OPT}}(f|\mathcal{S}) \propto a \log P_{\text{B}}(f|\mathcal{S}), \tag{6.1}$$

for some $a \approx 1$. Typically, $a \geq 1$ – i.e. the optimiser increases the bias towards the already high probability functions in $P_{\text{B}}(f|\mathcal{S})$. Is this always the case? What happens when we train more complex architectures on more complex data?

6.2.1 Feature learning

Fig. 6.2(a) shows a 2-layer CNN with pooling, trained on CIFAR-10. Despite a relatively small difference in generalisation error ($\langle \epsilon_G \rangle_{GP} = 83.9\%$ vs $\langle \epsilon_G \rangle_{OPT} = 85.3\%$), (a) clearly shows weaker correlation between $P_{\text{B}}(f|\mathcal{S})$ and $P_{\text{OPT}}(f|\mathcal{S})$ than seen in Fig. 6.1.

Fig. 6.2(b) shows the probability of each individual image being classified right or wrong (independently of all others). A necessary condition for two distributions to be identical ($P_1 = P_2$) is that the expectation of any function h must be the same under both, i.e., $\mathbb{E}_{f\sim P_1}[h(f)] = \mathbb{E}_{f\sim P_2}[h(f)]$. Fig. 6.2(b) clearly shows that for

some images, this criterion is not satisfied. Even if we relax the equality to allow for a log-linear relationship from Eq. (6.1), we can see that there is a difference – the two images in the bottom right corner are classified incorrectly $\sim 75\%$ of the time by the CNN-GP, but $\sim 5\%$ of the time by the CNN. This shows that the posterior mean itself is significantly different. Since these posterior marginals differ so significantly, the full posterior distributions $P_{\text{OPT}}(f|\mathcal{S})$ and $P_{\text{B}}(f|\mathcal{S})$ must be fundamentally different, suggesting that the optimiser is learning features not captured by the Bayesian prior.

The generalisation error is a coarse, one-dimensional summary of a model's performance. While both methods achieve a similar overall accuracy, Fig. 6.2(b) reveals that they do so by correctly classifying different subsets of the data. This figure plots the marginal probability of a correct classification for each individual test image – the posterior mean – for both $P_{\text{OPT}}(f|\mathcal{S})$ and $P_{\text{B}}(f|\mathcal{S})$. The lack of correlation shows that for many images, one model is confident in the correct label while the other is not.

6.2.2 Not actual feature learning

Fig. 6.2(c) uses the same chaotic neural networks from Chapter 4, but this time on MNIST. We use a 5-layer tanh-activated DNN with different initialisation scales $\sigma_w = 1, 3, 5$, trained with Adam (learning rate 0.001, batch size 128), compared to its NNGP equivalent. We only plot the probability of each individual image being classified correctly (rather than the full $P_{\text{B}}(f|\mathcal{S})$ vs $P_{\text{SGD}}(f|\mathcal{S})$ plot) because for $\sigma_w = 3, 5$ no function is found more than once in 10^5 samples. Fig. 6.2(c) shows that Adam acts like kernel ridge regression, rather than sampling from the posterior. Even the posterior mean of the highly chaotic ($\sigma_w = 5$) NNGP kernel would achieve 10% test error (compared to 41% for the expected individual sample). This shows that we can have significantly different generalisation errors without optimisers needing to do anything other than locate the posterior mean. How can this happen?

Let's consider the NNGP posterior. We stated the mean and variance for this case, assuming likelihood strength λ in Section 2.4 (Eqs. (2.14) and (2.15)),

which we restate below.

$$\mu(x_*) = k(x_*, S_X)^\top (K + (\lambda/\sigma_w)^2 I)^{-1} y$$
$$\sigma^2(x_*) = \sigma_w^2 \left(k(x_*, x_*) - k(x_*, S_X)^\top (K + (\lambda/\sigma_w) I)^{-1} k(S_X, x_*) \right).$$

As $\sigma_w \to \infty$, for fixed likelihood strength, $\sigma^2 \to \infty$. This means the signal-to-noise ratio approaches 0, as $k(x_*, x_*)$ is independent of σ_w. In essence, as we increase σ_w, the Gaussian $\mathcal{N}(\mu(x_*), \sigma_w^2(x_*))$ moves towards having equal probability weight on either side of the classification threshold. The mean will also shift. We can still recover the mean by taking many samples and taking the majority vote, but each individual prediction will have a greater probability of being wrong (assuming that $\mu(x_*)$ is correct). How can neural networks find a greater signal in the noise? As they are trained, gradients update large eigenvalue directions faster than smaller ones, creating a kind of "max-margin" effect, where noise dimensions disappear, effectively reducing the initialisation variance. See Bernstein et al. [115] (Figure 2) for a full explanation.

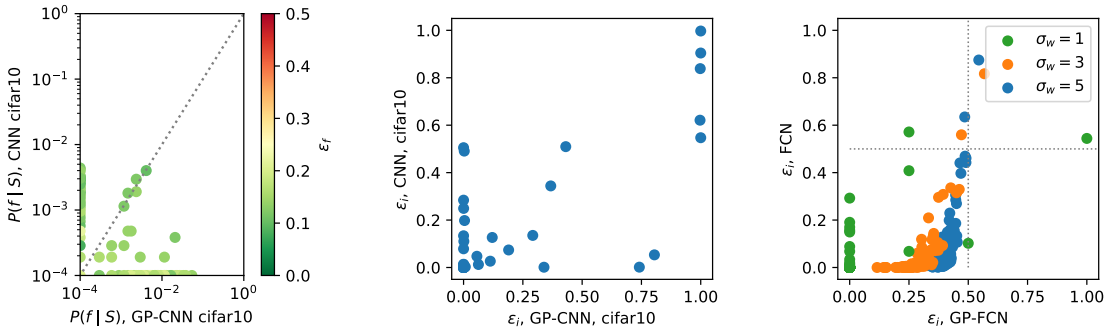


Figure 6.2: Extending the posterior comparison beyond the original FCN setting. (a) shows the same plot, but for the CNN in Fig. 6.1(f). $\langle \epsilon_G \rangle_{GP} = 83.9\%$, $\langle \epsilon_G \rangle_{OPT} = 85.3\%$. (b) shows the same data as (a), but presented differently: each datapoint shows the probability that each image is classified incorrectly. (c) shows a 5-layer FCN with width 1024 and tanh activations. We plot the probability of each image being incorrectly classified by the optimiser-trained neural network and its corresponding Gaussian Process. For $\sigma_w = 1$, $\langle \epsilon_G \rangle_{OPT} = 0.29\%$, $\langle \epsilon_G \rangle_{GP} = 0.23\%$; for $\sigma_w = 2$, $\langle \epsilon_G \rangle_{OPT} = 0.057\%$, $\langle \epsilon_G \rangle_{GP} = 29.4\%$, for $\sigma_w = 5$, $\langle \epsilon_G \rangle_{OPT} = 9.81\%$, $\langle \epsilon_G \rangle_{GP} = 41.0\%$. However, if we were to take the maximum a posteriori (i.e. the majority vote), both the GP and NN would predict the same classification except on one function (which the GP would get right and the NN wrong).

6.2.3 Remarks

We showed in Section 6.2.1 that a fine-grained analysis can show that feature learning can significantly alter $P_{\text{OPT}}(f|\mathcal{S})$ from $P_{\text{B}}(f|\mathcal{S})$. This demonstrates that $P(f)$ is not sufficient to understand generalisation.

But this is not unexpected: our toy model in Chapter 3 also had this behaviour once we used weight decay. But we were still sampling from a Bayesian posterior, and could understand exactly what the optimiser was doing – minimising $K_{DNF}(f)$.

Can we make similar claims for continuous neural networks trained with SGD? Having established that the NNGP is an insufficient model for SGD in these more complex regimes, the immediate next step is to better characterise the features that optimisers are actually learning.

6.3 Feature Learning: A Framework

In this section, we describe a framework for visualising and reasoning about feature learning. We define two qualitatively different feature learning regimes: the extended feature regime and the minimum feature regime, and show how optimiser hyperparameters and other architecture details lead to each.

Nam, Y., Mingard, C., Lee, S.H., Hayou, S. and Louis, A., 2024. Visualising Feature Learning in Deep Neural Networks by Diagonalizing the Forward Feature Map. arXiv preprint arXiv:2410.04264. [8].

Almost all deep neural networks (DNNs) can be simplified to a core structure: a highly expressive, learnable non-linear transformation (parameterised by weights w) followed by a final linear layer (parameterised by θ),

$$g(x) = \sum_{k=1}^{p} \theta_k \Phi_k(x; w), \qquad x \sim q, \quad \Phi : \mathcal{X} \to \mathbb{R}^p. \tag{6.2}$$

In other words, Deep Neural Networks (DNNs) are characterised by these *parameterised and learnable* feature maps Φ, which non-linearly transform the data space $\mathcal{X}$ into a representation $\mathbb{R}^p$. This is illustrated in Fig. 6.3(a). These representations correspond to the post-activations of the penultimate layer and serve as inputs

for the final linear layer. This is in contrast to a kernel method, or linear model, whose feature maps $\Phi_k(x)$ are *fixed* (Section 2.4).

Most DNNs share this fundamental abstract structure, primarily differing in how their feature maps are constructed. Examples of these feature map constructions include:

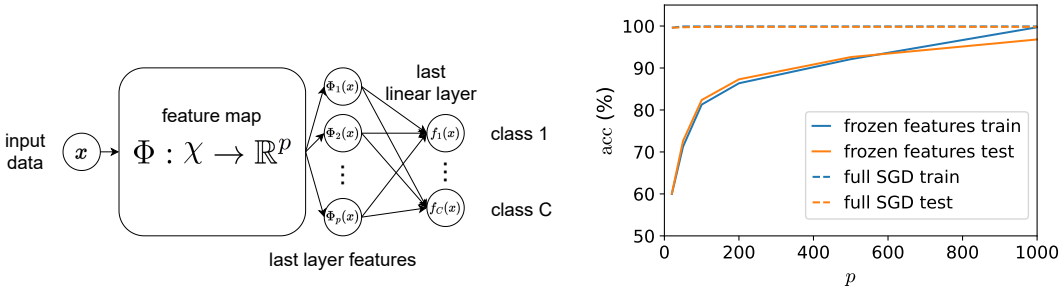
- A series of fully-connected layers, as seen in a Fully Connected Network (FCN).
- A combination of convolutional and pooling layers, characteristic of Convolutional Neural Networks (CNNs) [116], including very deep variants like VGG [117].
- A blend of convolutional layers and skip connections, as employed in ResNet [118].
- An encoder-decoder architecture, as utilized in the original Transformers [34], among many others.

From this simple decomposition, it follows that feature learning occurs when the forward feature map changes. More formally:

Definition 9 (Feature learning)**.** *A DNN is said to undergo feature learning if at any time t during training, $\Phi_{(t)} : \mathcal{X} \to \mathbb{R}^p$ differs from the forward feature map $\Phi_{(0)}$ at initialisation ($t = 0$). The feature maps are considered distinct if there exists an $x \in \mathcal{X}$ such that $\Phi_{(0)}(x) \neq \Phi_{(t)}(x)$.*

From this definition, it follows that if the feature map Φ remains fixed during training, no feature learning takes place. In this case, only the coefficients (parameters) of the final linear layer are adjusted during training, a scenario we define as follows:

Definition 10 (Coefficient Learning)**.** *A DNN is said to undergo coefficient learning if the forward feature map $\Phi_{(t)} : \mathcal{X} \to \mathbb{R}^p$ remains fixed at all times t during training such that $\forall x \in \mathcal{X}$, $\Phi_{(0)}(x) = \Phi_{(t)}(x)$. In this case, no feature learning occurs, and*



(a) Abstraction of DNN architecture

(b) CNN/MNIST performance by final layer width

Figure 6.3: A DNN can be decomposed into a feature map and a linear last layer of width p
(a) An abstract diagram depicting a DNN architecture as a combination of the forward feature-map $\Phi : \mathcal{X} \to \mathbb{R}^p$ from the input data space $\mathcal{X}$ to the p inputs of the final layer, and a final linear classifier for C-way classification. Within this picture, we define feature learning (Definition 9) as any change to the feature map Φ upon training, reflected in changes to the p 'features' compared to initialisation. An important aspect of modern DNN practice is that the final layer is typically small compared to the training set size n so that zero training error can only be reached by changing Φ, that is by feature learning. (b) Training (blue) and test error (orange) for a 5-layer CNN on the full MNIST dataset as a function of the width p of the final linear layer for the case of no feature learning, where the feature map is fixed (frozen) at initialisation (solid lines), and for standard SGD with backpropagation on all layers (dashed lines). For this simple example, one still needs a final layer of at least $p = 1,000$ to achieve zero training error with a fixed feature map, whereas if full feature learning is allowed, a width of just $p = 20$ is sufficient. For more complex datasets, much wider final widths, closer in size to the training set size n, would be needed for a random (frozen) feature map to achieve zero training error.

only the parameters (coefficients) of the final linear layer of the DNN are updated during training.

Coefficient learning occurs in the much-studied case of random features, where the parameters of Φ are set by a random sample over a distribution, and then not allowed to change under training. In the infinite width limit, it is equivalent to taking a sample from the NNGP [119]. We relate coefficient learning to the lazy regime in Section 6.3.4. We show the difference between pure coefficient learning and feature learning in Fig. 6.3(b), by comparing a frozen (finite) neural network with a fully trained network. When the frozen network is too narrow, the features required to fit the data and generalise are not present at initialisation. As the network width increases, eventually by random chance enough useful features are

present – at $p = 1000$, coefficient learning alone is enough to achieve 100% training accuracy and 95% test accuracy. For the network that is allowed to do both feature and coefficient learning, it achieves near 100% training and test accuracy at all widths. Note that feature learning clearly aids generalisation in this case, even when there are enough features to fit the data. See also Sections 2.6 and 2.7.

6.3.1 Features from the eigenfunctions of forward feature map

To define features, we begin by calculating the eigenfunctions $e_k : \mathcal{X} \to \mathbb{R}$ and eigenvalues ρ_k of Φ using a standard analysis from the kernel literature [120, 121] in terms of the integral operator $T : L^2(\mathcal{X}) \to L^2(\mathcal{X})$:

$$T[g](x') := \int_{\mathcal{X}} \Phi(x)^T \Phi(x') g(x) q(x) dx, \qquad T[e_k] = \rho_k e_k, \tag{6.3}$$

where $\langle e_i | e_j \rangle = \delta_{ij}$, and the bra-ket notation in $\langle \cdot | \cdot \rangle$ denotes the L^2 inner product defined on measure q (i.e. for two functions g, h $\langle g | h \rangle := \int_{\mathcal{X}} g(x) h(x) q(x) dx$). The integral operator T and its eigenfunctions $[e_1, e_2, \ldots, e_p]$ are of particular interest because T diagonalises the dynamics of the linear model trained with gradient descent (GD) and mean square error (MSE) loss[2]. Then the learning dynamics decompose into p independent dynamical equations, one for each eigenfunction:

$$g(x) = \sum_{k=1}^{p} \langle g | e_k \rangle \, e_k(x), \qquad \langle g | e_k \rangle \, (t) = \langle g^* | e_k \rangle \, (1 - e^{-\eta \rho_k t}), \tag{6.4}$$

Here, we have assumed that all coefficients $\langle g | e_k \rangle$ are initialised to zero at $t = 0$, and that the MSE loss is calculated with respect to a target function $g^*(x)$. Note that for fixed Φ, the speed at which each coefficient $\langle g | e_k \rangle$ is learned depends on the eigenvalues ρ_k and the learning rate η of the GD algorithm used. The larger the eigenvalue, the faster the coefficient is learned.

In the context of a DNN, the dynamics of Eq. (6.4) describe what would happen if one were to fix at a given time t', the hidden layers, and thus the feature map $\Phi_{(t')}(x)$

[2]Technically, these equations are obtained under a continuous limit of discrete-time GD dynamics, linearised to $\exp(-\eta \rho_k t)$. For completeness, we provide a full derivation of the dynamics of linear models in Appendix G.4

(with eigenfunctions e'_k), and subsequently train only the final layer. This picture is equivalent to interpreting the feature map at $t = t'$ as the one fixed at initialization so that there is no feature learning, and the DNN only learns the coefficients $\langle g^*|e'_k\rangle$ (although we might hope $\Phi_{(t')}(x)$ is better than $\Phi_{(t)}(x)$). Indeed, Eq. (6.4) describes the full dynamics of learning the coefficients for GD with MSE loss under the coefficient learning regime from Definition 10. We define features as follows:

Definition 11 (Features)**.** *Given the true data distribution q and a forward feature map $\Phi : \mathcal{X} \to \mathbb{R}^p$ mapping data to the input of the last layer of DNN, the p eigenfunctions $[e_1, e_2 \dots e_p]$ of the feature map are obtained by diagonalizing the integral operator T in Eq. (6.3). These eigenfunctions are indexed in descending order according to their corresponding eigenvalues ρ_k. The k^{th} feature is defined as a scalar-valued function on the input probability space, denoted as $e_k(x)$ for $x \sim q$. Since x is normally chosen at random, the features are scalar random variables.*

An intuitive way of thinking about this definition of the k^{th} feature is as a linear combination of the p entries of the input to the final layer, chosen such that it reflects the natural direction of the data after the transform via the feature-map Φ, similar in spirit to how SVD finds the "natural direction" of the matrix. Variations of the definition above are also possible, see Appendix G.3. See Appendix G.1 for details on practical calculations.

6.3.2 Measuring feature learning by projecting onto the learned function and the target function

While using Definition 11 for features has certain practical advantages, which we will explore further in this paper, the eigenfunctions (features) are typically hard to visualise directly for high-dimensional problems. To make progress in analysing and visualising feature learning, we will define measures based on projecting the eigenfunctions onto either the target function g^*, which describes the target function on all the data $x \in \mathcal{X}$, or else the learned function $\hat{g}$ that the DNN expresses once training on a training set $S_{tr} \in \mathcal{X}$ is complete.

Before defining our measures, we note that for the sake of simplicity, we considered DNNs with scalar output functions g in the previous sections. For a balanced C-way classification task, the target function is a vector function (i.e. $g^* : \mathcal{X} \to \mathbb{R}^C$), and the dataspace $\mathcal{X}$ can be partitioned into subspaces $\{A_1, \cdots, A_C\}$ by class. The components $g_i^* : \mathcal{X} \to \mathbb{R}$ for $i \in [1, \cdots, C]$ of $g^* = [g_1^*, \cdots, g_C^*]$ can be written down as:

$$g_i^*(x) = \mathbf{1}_{A_i}(x) \qquad (A_i = \{x : x \text{ is in class } i\}) \tag{6.5}$$

where $\mathbf{1}_{A_i}$ is the indicator function for A_i. Note that because all the classes are mutually exclusive, we have

$$\langle g_i^* | g_j^* \rangle = \int_{\mathcal{X}} g_i^*(x) g_j^*(x) q(x) dx = \frac{1}{C} \delta_{ij}, \tag{6.6}$$

where the C^{-1} on the right-hand side comes from the probability measure for each class i. Then it follows that at least C orthogonal scalar output functions are needed to express g^*. We define the target function space $\mathcal{H}^*$ as the function space spanned by $[g_1^*, \cdots, g_C^*]$.

Given a C-class vector target function g^* and a projection operator $P_{\mathcal{H}^*} : L^2(\mathcal{X}) \to L^2(\mathcal{X})$ onto the target function space $\mathcal{H}^*$, we define the *quality* Q_k^* of a feature e_k as follows:

$$P_{\mathcal{H}^*}[g] := \sum_{i=1}^{C} \frac{1}{\|g_i^*\|^2} |g_i^*\rangle \langle g_i^* | g \rangle , \qquad Q_k^* := \frac{\langle e_k | P_{\mathcal{H}^*}[e_k] \rangle}{C} = \sum_{i=1}^{C} \langle e_k | g_i^* \rangle^2 . \tag{6.7}$$

The range of Q_k^* is given by $0 \leq Q_k^* \leq C^{-1}$. We use the word quality because Q_k^* measures how much e_k overlaps with the C dimensional target function space $\mathcal{H}^*$. The higher the quality, the more of the target function is captured by the feature. We also define a cumulative measure of the quality of the first k eigenfunctions, ordered by the size of their eigenvalues:

$$\Pi^*(k) := \sum_{j=1}^{k} Q_j^* = \sum_{j=1}^{k} \sum_{i=1}^{C} \langle e_j | g_i^* \rangle^2 . \tag{6.8}$$

We will call this measure the *cumulative quality*. Its range is given by $0 \leq \Pi^*(k) \leq 1$. Assuming the g_i^* are linearly independent, $\Pi_g^*(k)$ measures how much of the C

dimensional function space spanned by g is captured by the first k eigenfunctions or features. Another way of expressing this concept is to say that $\Pi^*(k)$ measures how well the first k eigenfunctions are *aligned* with g^*. For a DNN with a final layer width p, there are p features (eigenfunctions). We expect that *total alignment* or *total quality* of all the features, $\Pi^*(p)$, will typically increase under training. The highest possible total alignment/quality, when the DNN expresses the true-function g^* is $\Pi^*(p) = 1$. The minimum number of features for which $\Pi^*(k) = 1$ can be attained is at $k = C$, which would occur when the quality of Q_k of each feature is at its maximum value of $1/C$. Related cumulative measures of alignment/quality in the context of deep learning can be found, for example, in [122, 123, 98, 124]. In particular, recent studies [98, 125, 100, 99, 126] have linked better alignment at lower k to better generalisation performance in overparameterized linear models. Due to the narrow final layer widths, the learning problem described here is underparameterized, but the idea that good generalisation correlates with large $\Pi^*(p)$ should be robust.

Analogous measures to those above for the target function g^* can also be defined for the vector function $\hat{g} : \chi \to \mathbb{R}^C$ expressed by the DNN after training, which we call the 'learned function'. The learned function space is given by $\hat{\mathcal{H}} = span\{\hat{g}_1, \cdots, \hat{g}_C\}$. Similarly, a projection operator $P_{\hat{\mathcal{H}}}$ onto the learned function space can be defined as in Eq. (6.7). This can then be used to define the *utility* $\hat{Q}_k$ of the kth feature, which measures how much the trained DNN uses that feature to express $\hat{g}$:

$$\hat{Q}_k = \frac{\langle e_k | P_{\hat{\mathcal{H}}}[e_k] \rangle}{\dim(\hat{\mathcal{H}})}. \tag{6.9}$$

Similarly, we can define a cumulative measure of the utility of the first k features, ordered by the size of their eigenvalues:

$$\hat{\Pi}(k) = \sum_{j=1}^{k} \hat{Q}_k \tag{6.10}$$

which we will call the *cumulative utility*. It takes values $0 \leq \hat{\Pi}(k) \leq 1$. Note that typically, $\dim(\hat{\mathcal{H}}) = C$, since the entries of $\hat{g}$ are likely to be linearly independent. Additionally, the entries of the output function typically exhibit similar norms

after training. (i.e. $\|\hat{g}_i\| \approx \|\hat{g}_j\|$). By definition, the *total utility* $\hat{\Pi}(p) = 1$. The smallest k for which $\hat{\Pi(p)} \approx 1$ provides a measure for the effective number of features used by the DNN. By contrast, the total cumulative quality $\Pi^*(p)$ can still be much smaller than 1 if the training leads to a function $\hat{g}$ that significantly differs from the ground truth g^*.

6.3.3 Eigenvalues and effective dimensions

The eigenvalues determine the speed of coefficient learning under MSE loss in a linear model, as shown in Eq. (6.4). They can also be written in the following way:

$$\rho_k = \langle e_k \,|\, T[e_k] \rangle = \sum_{j=1}^{p} \langle \Phi_j \,|\, e_k \rangle^2 \,. \tag{6.11}$$

which shows that they are a measure of the projection of Φ along e_k in function space. It is therefore of interest to plot the distribution of the p eigenvalues as a function of index k.

It can also be useful to define scalar measures to characterise these distributions. For any vector a of positive numbers a_i, the exponential of Shannon entropy [127] can be used as a spectral dimension or effective dimension measure:

$$D_{\text{eff}}(a) = \exp\left(-\sum_i \frac{a_i}{\sum_j a_j} \ln\left(\frac{a_i}{\sum_j a_j} \right) \right). \tag{6.12}$$

If the vector a has d components, then $D_{\text{eff}}(a)$ has a maximal value of d when all components are equal (and non-zero). If the distribution of its components is non-uniform, then the effective dimension will be lower than d, with a minimum of 1 if a only has one non-zero component.

The effective dimension or effective rank of T can be measured via $D_{\text{eff}}(\rho)$ where $\rho = [\rho_1, \cdots, \rho_p]$. In a way that is analogous to the way that entropy quantifies the average number of bits for a symbol in the source coding theorem, this measure implies that although T has a rank of p, it can in principle be compressed to an operator of rank $D_{\text{eff}}(\rho) \leq p$. From the perspective of Φ, $D_{\text{eff}}(\rho)$ measures how many e_k you need in expectation to describe $\Phi(x)$ for $x \sim q$. As a practical note, because the first eigenfunction of T upon full training is typically a constant, as discussed in

Appendix G.6, $D_{\text{eff}}(\rho)$ in our study is computed without the first eigenvalue before reintroducing a single dimension (i.e. $D_{\text{eff}}(\rho) := 1 + D_{\text{eff}}([\rho_2, \cdots, \rho_p])$).

We can also use Eq. (6.12) to measure the effective dimension $D_{\text{eff}}(Q^*)$ of the set of feature qualities $Q^* = [Q_1^*, \cdots, Q_p^*]$. This provides a measure of the effective number of features needed to express the target function. Because the target function may not always be completely expressible by features, this measure can be interpreted as the effective number of features required for the best possible approximation of the target function, given Φ.

Similarly, we can define $D_{\text{eff}}(\hat{Q})$ for the feature utilities $\hat{Q} = [\hat{Q}_1, \cdots, \hat{Q}_p]$, which describes the effective number of features used to describe the learned function. We expect that $D_{\text{eff}}(\hat{Q})$ is roughly the number of features needed to achieve $\hat{\Pi}(k) \approx 1$, which is one measure of the effective number of features used by the DNN to express the learned function $\hat{g}$.

6.3.4 The minimum feature (MF) regime and the extended feature (EF) regime

In the simple examples shown in Figs. 6.4 and 6.6, a DNN trained with SGD converges on a solution where the number of used features and the number of significant eigenvalues are both close to C. In other words, $D_{\text{eff}}(\hat{Q}) \approx C$ and $D_{\text{eff}}(\rho) \approx C$ respectively. For the C-way balanced classification task, we will say that a DNN is in the "minimum feature (MF) regime" if the following qualitative properties are satisfied:

1. $\hat{\Pi}_{\hat{g}}(k)$ increases linearly until $k = C$ with $\hat{\Pi}(C) \approx 1$. ($D_{\text{eff}}(\hat{Q}) \approx C$)).
2. The first eigenfunction e_1 (which is a constant function, see Appendix G.6 for further discussion) has an eigenvalue of $a_1 > 0$, while the next $(C-1)$-eigenfunctions $e_2, \cdots, e_C$ all have eigenvalues of $a_2 > 0$. In addition, all other eigenvalues decrease significantly for $k > C$ ($D_{\text{eff}}(\rho) \approx C$).

It would also be useful to derive a quantitative scalar measure of the MF regime. To that end, we first define the state of the operator T in which the above conditions are satisfied exactly.

Definition 12 (Minimum Projection (MP) operator)**.** *For a DNN on a balanced C-way classification task, the integral operator T is an MP-operator T_{MP} if it has two nontrivial eigenspaces, one being the span of constant function $\mathbf{1}$ and the other orthogonal complement of $\mathbf{1}$ inside $\hat{\mathcal{H}}$, where $\hat{\mathcal{H}}$ is C-dimensional function space spanned by entries of the learned function $\hat{g} : \mathcal{X} \to \mathbb{R}^C$ expressed by the DNN. This is equivalent to that T_{MP} is given as*

$$T_{MP}[u] \;=\; a_1 \,\langle \mathbf{1} \,|\, u \rangle \;\mathbf{1} \;+\; a_2 \, P_{\hat{\mathcal{H}} \cap \mathbf{1}^\perp}(u), \tag{6.13}$$

where a_1, a_2 are positive scalars, and $P_{\hat{\mathcal{H}}}$ denotes the orthogonal projection onto $\hat{\mathcal{H}}$.

Ignoring the constant function, whose discussion is deferred to Appendix G.6, the MP-operator is a simple projection onto $\hat{\mathcal{H}}$. It is easy to see that the operator $P_{\hat{\mathcal{H}}}$ satisfies the qualitative properties above. First, $P_{\hat{\mathcal{H}}}$ has only C eigenfunctions, and $\hat{Q} = C^{-1}$ by Eq. (6.10), satisfying the first property[3]. The second property follows since $P_{\hat{\mathcal{H}}}$ has C equal non-zero eigenvalues. We note that the T_{MP} operator is closely related to the phenomenon of NC [128].

Having defined T_{MP}, we will use the centered kernel alignment (CKA) measure (see e.g. [129]) to calculate how close an empirically measured T is to the idealised operator T_{MP}:

$$CKA(T, T_{MP}) = \frac{\mathrm{Tr}(c(T)c(T_{MP}))}{\|c(T)\|_F \|c(T_{MP})\|_F}, \tag{6.14}$$

where $c(A)$ is centering operator $(I - |\mathbf{1}\rangle \langle \mathbf{1}|)A(I - |\mathbf{1}\rangle \langle \mathbf{1}|)$, I is identity and $|\mathbf{1}\rangle$ is the constant function 1 in $L^2(\mathcal{X})$, and $\| \cdot \|_F$ is the Frobenius norm. CKA has been used for comparing features of DNNs [129], and also for studying the evolution of the NTK [130, 3]. Notably, it has the advantage that the measure is invariant to isotropic scaling.

The CKA measure allows us to define a scalar quantitative criterion to measure whether a DNN is in the MF regime, which we can use as an additional layer of assessment to complement our more qualitative definitions 1. and 2. above:

[3] This assumes that $[\hat{g}_1, \cdots, \hat{g}_C]$ are C linearly independent, which is typically true for DNN trained on a balanced dataset.

Definition 13 (CKA Minimum Feature Regime Measure κ_{CKA})**.** *For a distribution q and a class-balanced learned function $\hat{g}$, a DNN is in the MF regime if $\kappa_{CKA} = 1 - \text{CKA}(T, T_{MP}) < \epsilon$, where T is the feature kernel (operator) of a DNN and T_{MP} is the MP-operator.*

The advantage of this scalar measure is that it gives a single number that can be reproducibly used to distinguish regimes. There is some arbitrariness with the choice of ϵ, but we find that $\epsilon = 0.1$ correlates well with more informal visual measures of the MF regime using the criteria above. Needless to say, these measures do not exhaust all the information that can be gleaned from observing the eigenvalue distributions and the two projection measures as a function of k.

We can also define an **extended feature (EF) regime** for the case where, on the one hand $\kappa_{CKA} > 0.1$, and on the other hand the coefficient learning regime has not been reached so that $\Phi_{(t)}(x) \neq \Phi_{(0)}(x)$ for at least one $x \in \mathcal{X}$. Therefore, the EF regime encompasses everything between strict coefficient learning and the MF regime. We note that it could be interesting to also define a scalar measure similar to κ_{CKA} of how far the EF regime is from the strict coefficient learning regime, but we leave that for future work. We have also focused on balanced C-class classification. It would also be interesting to extend these definitions to datasets that are not class-balanced, which we leave to future work.

6.3.5 Architectures and Datasets used in experiments

In this section, we mainly confine ourselves to classification on standard datasets such as MNIST, CIFAR-10, and CIFAR-100, and use MSE loss. Where not explicitly stated, we use the SGD with momentum of 0.9, weight decay of 10^{-3}, and a learning rate of 0.05, which is decayed by a factor of 0.2 every 60 epochs. For CIFAR-10, all models are trained for 200 epochs, in which both the training and test loss converged, using standard data augmentation (random crops and horizontal flips). For CIFAR-100, all models are trained for 600 epochs without a learning rate scheduler, using standard data augmentation (random crops and horizontal flips).

The training accuracy is over 99.5% for all models unless explicitly stated. For further details on the experimental setups, see Appendix G.7.

6.4 Feature Learning: Results

A toy example Fig. 6.4 shows a 1-dimensional toy example of a 4-layer FCN learning a Heaviside function with SGD and MSE loss. The advantage of this simple system is that the eigenfunctions can be directly visualised. In sub-figures (a–c), we show the first three eigenfunctions for epochs 0, 10, and 100, respectively. At epoch 100, the MSE loss is less than 10^{-5}, and we consider training to have converged. Note that the rather dramatic feature learning is reflected in changes to the eigenfunctions (thus features) during training of the DNN with SGD.

Interestingly, a similar MSE loss can quite easily be achieved without any feature learning by using fixed hidden layers, e.g. a fixed parameter-function map. However, when we train with full SGD, the DNN does not converge to this coefficient learning solution, which needs on the order of 100 eigenfunctions to reach our desired loss value. Instead, the DNN changes the feature map until only one eigenfunction, the one with the biggest eigenvalue, dominates the fit to the target function.

To illustrate how our projection measures $\Pi^*(k), \hat{\Pi}(k)$ and the eigenvalue distributions ρ_k can be used to monitor and visualise feature learning, we plot them in Fig. 6.4(d-f), at 0, 10, and 100 epochs. For example, $\Pi^*(k)$ provides a 1-dimensional visualisation of the observation from Fig. 6.4(a-c) that fewer eigenfunctions are needed in describing the target function as training progresses. This reduction in the number of required eigenfunctions is also reflected in $D_{\text{eff}}(Q^*)$, which decreases from 3.2 at epoch 0 to 1.1 at epoch 200. Similarly, $\hat{\Pi}(k)$ helps visualise how many eigenfunctions were used to describe the learned function with $D_{\text{eff}}(\hat{Q})$ decreasing from 1.8 to 1.0 after training. Finally, the normalized eigenvalues ρ_k/ρ_1 exhibit faster decay after training, or equivalently a decrease in the effective number of features, as measured by $D_{\text{eff}}(\rho)$, which is 1.6 at initialization but 1.2 after training[4].

[4]Note that this is the only case where we do not make exceptions regarding the first eigenvalue

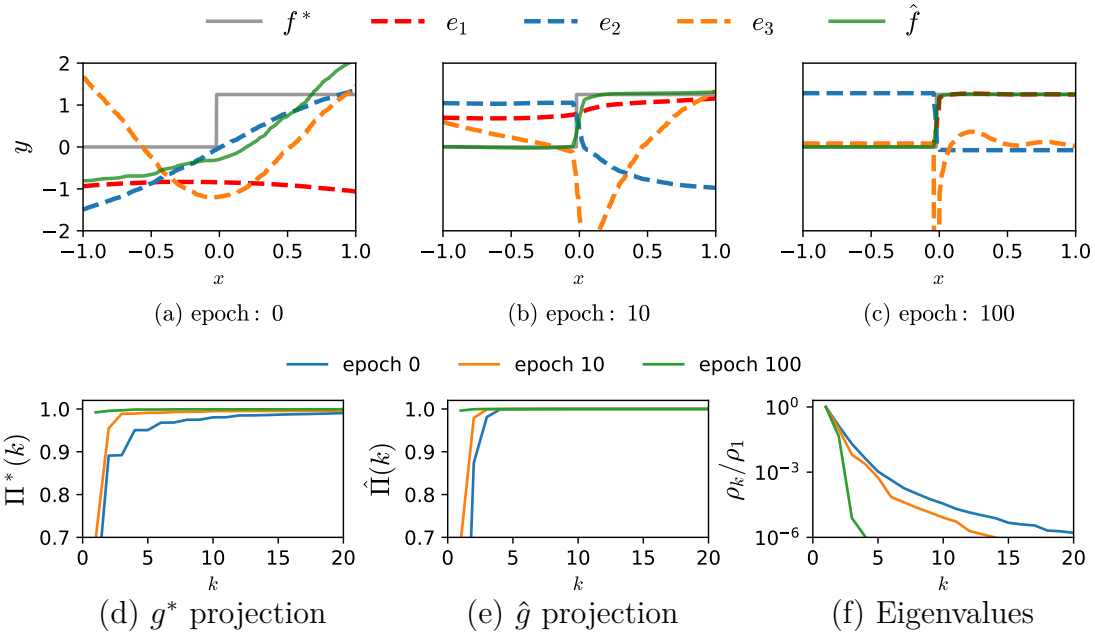


Figure 6.4: Toy model demonstrating feature learning by a DNN A 4-layer FCN with width 1000 and scalar input and output, is trained to learn the Heaviside step function g^* over the domain $[-1, 1]$. (a) At initialisation (epoch 0), the first three eigenfunctions e_i (dashed) of the feature map Φ are shown in order of their eigenvalues. If Φ and thus the features are fixed, about 20 features are needed to accurately approximate the target function g^* (solid grey) to an MSE loss less than 10^{-5}. (b,c) During training with SGD, the DNN learns new features with better quality Q_K^*, greatly reducing the number needed to express the target function g^* and the learned function $\hat{g}$. To visualise how feature learning develops with epochs during training, the target function g^* (d), and the learned function $\hat{g}$ (e), are projected onto the first k eigenfunctions using Eq. (6.8) and Eq. (6.10) respectively, illustrating that fewer features are needed after training. (f) The eigenvalues ρ_i decay significantly faster post-training.

Kernel methods vs Feature learning In Fig. 6.5(a), we show how our measures change as a function of training-set size for our standard implementation of ResNet18 on CIFAR-10 (details in Section 6.3.5). For smaller training sets, the system is clearly in the EF regime, as can be seen, for example, by the difference between the cumulative quality at $k = C$, $\Pi^*(k = 10)$, and the final alignment/quality $\Pi^*(p)$, by the fact that the eigenvalue distribution does not have a clear plateau for $1 < k \leq C = 10$, and by the scalar κ_{CKA} measure. There is clearly a transition from the EF regime to the MF regime between $n = 1000$ and $n = 5000$, as can be seen by examining these same measures.

when calculating $D_{\text{eff}}(\rho)$.

For this ResNet18, the final layer width is $p = 512$. If the amount of training data $n \lesssim p$, a DNN should be able to reach zero training error by coefficient learning (Definition 10). For larger training set sizes, feature learning will be a precondition for reaching zero training error due to the limited expressivity of the random feature model for such a relatively narrow last layer where $n \gg p$ (see also Appendix G.2). Coefficient learning alone will not be enough. While more work is needed to understand exactly why a DNN converges to the EF or the MF regime, it may not be surprising that a DNN will use more features if the system is closer to a regime where it could also achieve zero training error with only coefficient learning.

There has been a lot of recent interest in scaling laws for DNNs. These include how generalisation performance as a function of model size, training set size, or simply the cost of computational resources, see e.g. [131, 132, 88, 133, 63, 134, 6] and references therein. It is interesting to ask what the impact of feature learning is on these scaling laws. To that end, we plot, in Fig. 6.5(b), the learning curves for the test accuracy (i) and test loss (ii) for the ResNet18 from Fig. 6.5(a). We also show the learning curves for the NNGP corresponding to the same ResNet18 architecture. Because it is computationally difficult to use the NNGP with augmented data, we also include the ResNet18 for non-augmented data for a more direct comparison to the NNGP. The NNGP will, by definition, not feature-learn, so comparing the two machine-learning methods can shed light on how feature learning affects performance as a function of training set size n. For the NNGP, we observe a scaling law with close to a fixed exponent over the domain. For the two ResNet18 variants, we find that after an initial lower exponent, there is a clear transition from a slower decay rate in the EF regime, similar to that of the NNGP, to a faster decay with a larger exponent once full feature learning to the MF regime sets in.

Interestingly, prior work has also shown that kernel regime (infinite-width DNNs with NTK or NNGP parameterisation) has a smaller exponent in the decay of learning curves, see e.g. [135, 136]. Here we explicitly show that the change in scaling exponent correlates with a change in feature learning. Whether this correlation is causal is left for future work. As can be seen in Fig. 6.5(b)(iii), the κ_{CKA} measure

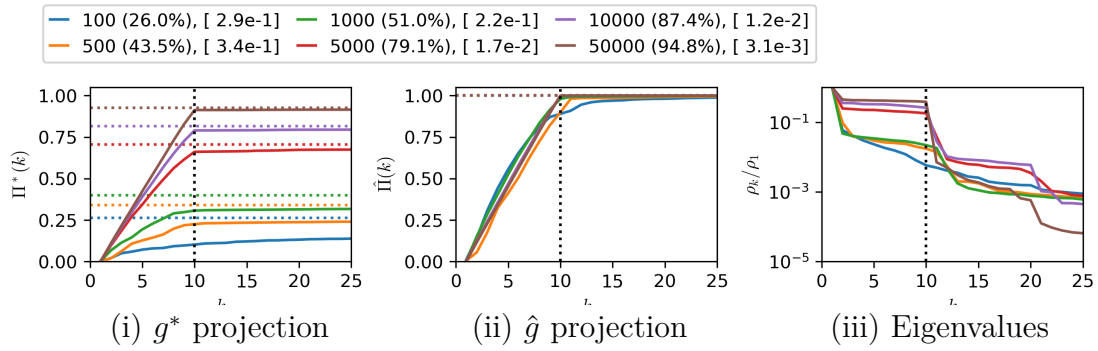
shows that the MF regime occurs for smaller n for the ResNet18 with augmented data, than for the case of no data augmentation. The point at which $\kappa_{CKA} \lesssim 0.1$ is also roughly the point at which a different higher exponent kicks in for the learning curves. Both learning curves have very similar exponents in the MF regime. Prior results comparing the scaling exponents from marginal-likelihood PAC-Bayes bound to full DNN calculations in [63] suggest that the effect of feature learning on the decay rate will be larger for more complex datasets. More work needs to be done to fully understand the effect of feature learning on scaling exponents.

Learning rate and features In Fig. 6.6 we explore two examples of how hyperparameter tuning affects performance and feature learning. One of the most common hyperparameters to tune is the learning rate. As expected, we find that performance is affected by the learning rate. In Fig. 6.6(a), we observe that the two better-performing learning rates are in the MF regime. For the smallest learning rate, which also performs significantly less well, we observe the EF regime.

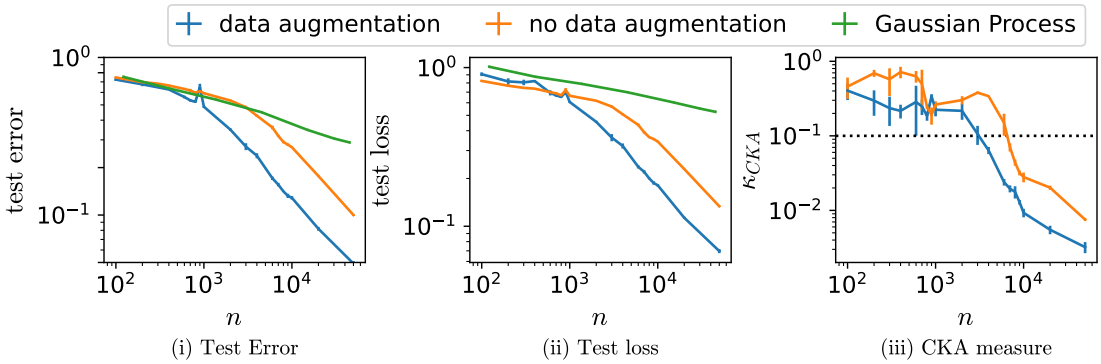
In Fig. 6.6(b), we show that weight decay also flattens the distribution of the first C eigenvalues. It has been argued in [137] that weight decay contributes to finding a low-rank solution. However, we note that here the MF regime is also achieved without weight decay.

Layerwise training & Batch norm Intermediate layers have an intermediate feature map in the same way the last layer does. The feature map of the l'th layer is $\Phi^{(l)} : \mathcal{X} \to \mathbb{R}^{p_l}$, where p_l is the number of features for layer l. This induces a feature kernel for the l'th layer, $\Phi^{(l)}(x)^T\Phi^{(l)}(x')$. Its eigenvalues, eigenfunctions and all projection measures in Section 6.3 can be defined in exactly the same way that they are for the last layer. When the output of the feature map $\Phi^{(l)}$ is multidimensional (e.g. the output from a 2D convolutional module), the output is flattened to a vector (similarly to [138, 139, 123]).

The interpretation of $\Phi^{(l)}$ as an intermediate feature map and measuring its change with training to monitor feature learning in intermediate layers is straightforward. We use the measures Π^* and $\hat{\Pi}$, which capture how aligned a layer



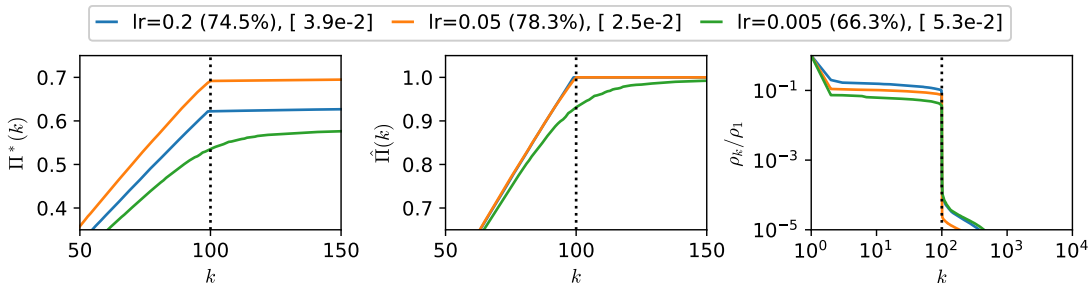
(a) Feature learning as a function of training set size



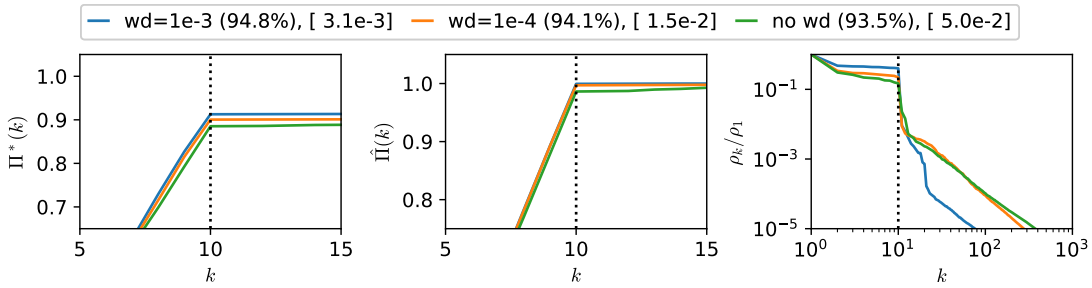
(b) Learning curves for variants of ResNet18 on CIFAR-10

Figure 6.5: Feature learning as a function of training set size n. a) For a ResNet18 on CIFAR-10 using our standard hyperparameters with data augmentation (Section 6.3.5) we show (i) the cumulative quality (Eq. (6.8)), horizontal dotted lines denote the final quality/alignment $\Pi^*(p)$; (ii) the cumulative utility (Eq. (6.10)) and (iii) the eigenvalue distribution (Eq. (6.3)) with a vertical dotted line at $k = C$. Above the figure, the test accuracy is shown in curved brackets, while the CKA measure κ_{CKA} (13) is shown in square brackets. The MF regime is defined to occur when $\kappa_{CKA} \leq 0.1$. Both the final quality/alignment $\Pi^*(p)$ and the generalisation performance increase with increasing training size n, as expected. Between training set sizes $n = 1000$ and $n = 5000$ there is a clear transition from the EF regime to the MF regime, observable both in κ_{CKA} and in the eigenvalue distribution. (b) Shows the associated learning curves for (i) the test error, and (ii) the MSE loss for the ResNet18 model used in (a), and also for an NNGP for the same ResNet18 architecture which has no feature learning, and the ResNet18 model without data augmentation that matches the NNGP setup. The two feature learning models show a marked shift in the rate of decrease of test error and test loss accompanied by a drop in κ_{CKA} at roughly the same values of m, as shown in (iii). The NNGP shows a more constant decay rate in error and loss.

is to the target function and learned function, respectively. It is straightforward to simply consider these as probes that help quantify the amount of feature learning that occurs under training. However, more care is needed in their direct interpretation. In contrast to the final-layer features, these layers are not used to directly produce



(a) ResNet18 on CIFAR-100 with different learning rates



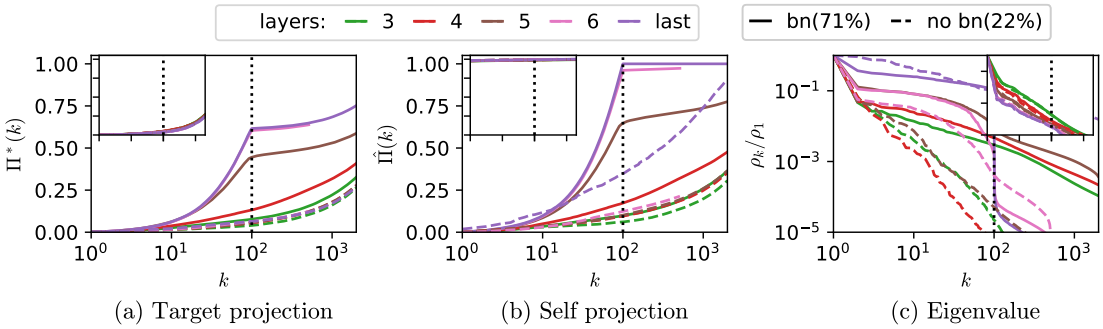
(b) ResNet18 on CIFAR-10 with different weight decay rates

Figure 6.6: Effects of learning rates and weight decay on feature learning and generalization In (a) we explore the effect of changing the learning rate for a ResNet18 on CIFAR-100. Note that the lower learning rate leads to the EF regime, and poorer performance. The best performance is for the intermediate learning rate, which is in the MF regime. The higher learning rate, with slightly worse performance, is also in the MF regime. In (b) we study the effect of weight decay for a ResNet18 on CIFAR-10 where we observe that a tighter MF regime correlates with a larger weight decay and better performance.

measured output, but rather are fed into the next layer. If well-trained early layers are strongly aligned to the final target, that may not be a good thing. We illustrate this in Fig. 6.9 where a greedy layerwise training method leads to stronger alignment in earlier layers, but worse performance than normal SGD with backpropagation for a ResNet18 on CIFAR-10.

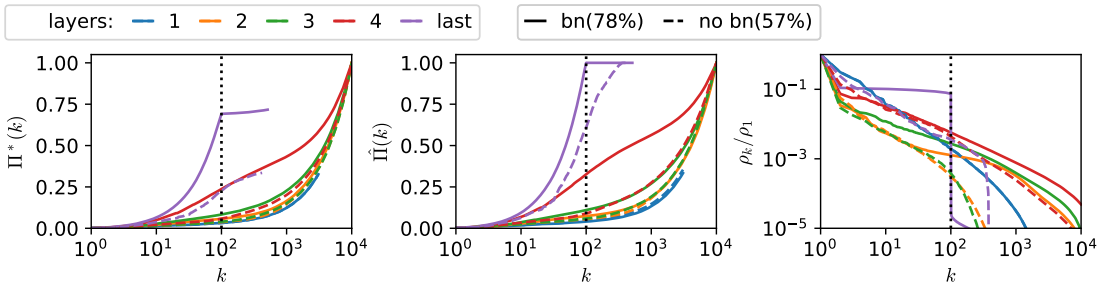
We also study the effect of batchnorm on the intermediate layer features in Fig. 6.7 for two cases where it makes a big difference to generalisation performance. Subfigure (a) shows VGG16 on CIFAR-100 and (b) ResNet18 on CIFAR-100. In both cases, early layers barely align to the target function without batch norm and perform significantly worse than the case with batch norm. This provides an example of how intermediate layers help diagnose the causes of the reduced

performance. We leave further experiments and analysis to future work, including an upcoming companion paper on intermediate features.



(a) Target projection (b) Self projection (c) Eigenvalue

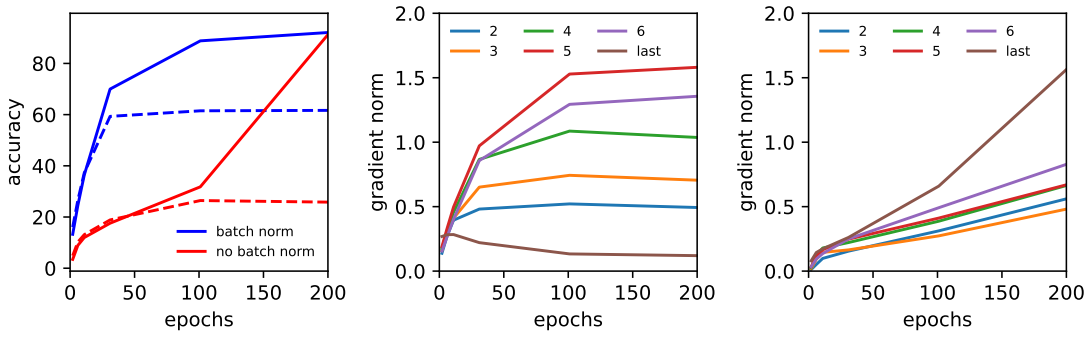
(a) VGG16 CIFAR-100



(i) Target projection (ii) Self projection (iii) Eigenvalue

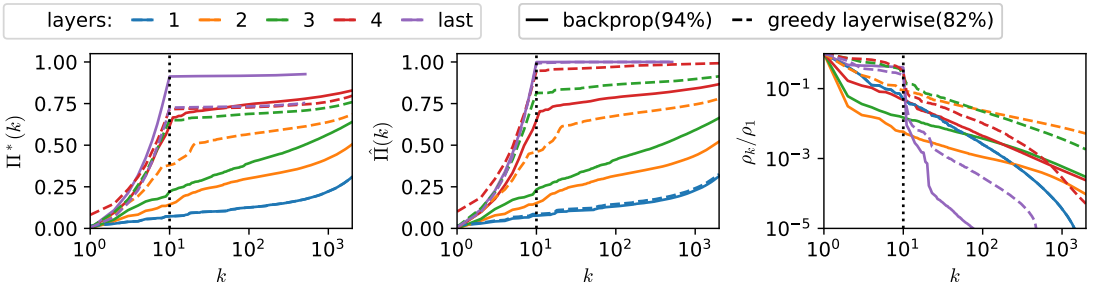
(b) WideResNet18 CIFAR-100

Figure 6.7: Effect of batch normalization in CIFAR-100. We clearly observe more alignment in the intermediate layers, and the performance boost is more dramatic (22% to 71% for (a), 57% to 78% for (b)). Moreover, without batch normalization, neither model achieved over 99% training accuracy in 600 epochs, unlike other experiments in the paper. In this case, training was instead halted at 94% after 1200 epochs. (c) The key distinction lies in the eigenvalues, where the decay rate was slowed down after the application of batch normalization.



(a) accuracy (b) batch norm (c) no batch norm

Figure 6.8: Gradient norm for Fig. 6.7(a). (a) The solid and dotted lines represent the training and testing accuracies, respectively, for VGG16 on CIFAR-100 with and without batch norm. (b) and (c) plot the gradient norms over the training epochs for the VGG16 with batch norm and without it, respectively. By using batch norm, as shown in (b), significantly larger gradients flow through the intermediate layers compared to the model without batch norm (c). This increased gradient flow leads to greater feature learning (as shown in Fig. 6.7) and leads to a substantial improvement in the test accuracy, seen in (a). Note that test accuracy we reach in the batch norm case is 62%, less than the 71% quoted in Fig. 6.7(a), with the discrepancy being a result of training time (200 epochs vs 1200).



(a) g^* projection (b) $\hat{g}$ projection (c) Eigenvalues

Figure 6.9: Greedy layerwise training leads to a collapse of features in earlier layers. In contrast to full backpropagation (solid), greedy layerwise training (dashed) leads to more pronounced alignment at lower k for earlier layers, as seen, for example in the value of $\Pi^*(C)$ for layers 2 and 3 in (a). However, there is only a minute enhancement in $\Pi^*(k)$ for subsequent layers for greedy layerwise training, leading to final poorer quality features compared to that of full backpropagation. Note that the difference in $\Pi^*(C)$ for the final layer directly correlates with the performance as both models are in a tight MF regime as seen in (b). Note that $\hat{Pi}(k)$ of the last layer is almost identical for both models (purple solid and dashed lines in (b)). Similarly, at layers 3 and 4, the intermediate layer eigenvalues for greedy layerwise training in (c) are more MF regime-like with a larger gap between ρ_C and ρ_{C+1}. This is quantified in $D_{eff}(\rho)$ as 348 vs. 608 for layer 3 and 90.4 vs. 279 for layer 4 for layerwise training and full backpropagation, respectively.

6.5 Remarks

In this section, we will use discussion points from several papers I made secondary contributions to. The first authors were responsible for the overall idea, the bulk of the theory and the majority of the experiments. The references themselves are placed just before the sections they contribute to.

In Chapters 4 and 5, we argued that the prior, $P(f)$, is a key driver of generalisation. Here, we connect this idea to the empirical findings of the current chapter, which demonstrate that neural networks can depart from this initial bias through feature learning. To do this, we will address the following two questions:

Q(i) Given a neural network with a prior over parameters $P(\theta)$, some data S, and a likelihood function $L(S;\theta)$, how accurately does the Bayesian posterior

$$P(f|S) = \int d\theta P(\theta|S)\mathbb{1}[f(\theta) = f] \tag{6.15}$$

$$P(\theta|S) = \frac{L(S;\theta)P(\theta)}{\int d\theta L(S;\theta)P(\theta)} \tag{6.16}$$

approximate the posterior induced by an optimiser, P_{opt}?

Q(ii) Given that the answer to Q(i) is well-understood for networks that do not feature learn (such as those with frozen layers or infinite-width networks under NTK or SP parameterisation), what happens to the feature embeddings, $\Phi_k(x;\theta)$, when feature learning does occur?

6.5.1 Near the no-feature-learning regime

In Section 6.1, we showed that the optimiser posterior, $P_{\text{OPT}}(f|\mathcal{S})$, of very wide neural networks trained with MSE loss on simple tasks was highly correlated with the Bayesian posterior, $P_{\text{B}}(f|\mathcal{S})$, for the corresponding NNGP using a square likelihood. This alignment is theoretically guaranteed under two conditions:

- Training a neural network with all but the last layer frozen. In the infinite-width limit, the feature maps $\Phi_k(x)$ are equivalent to the features of the NNGP and remain fixed during training.

- Training an infinite-width network under SP with an infinitesimal learning rate yields lazy/NTK dynamics in which all layers move only infinitesimally; the induced kernel stays (approximately) constant and features remain close to initialisation.

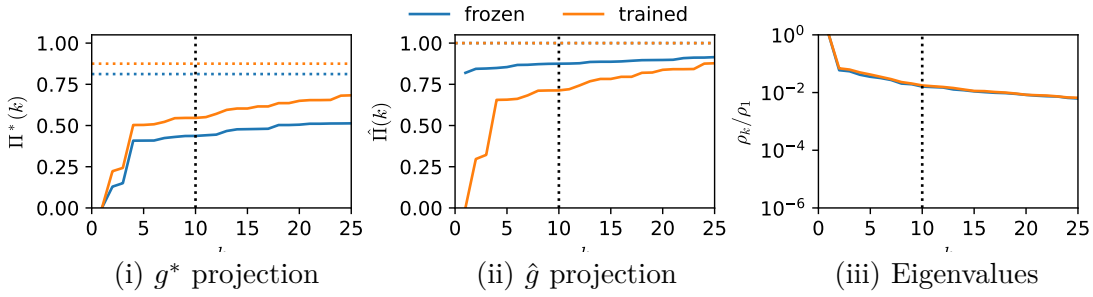
These conditions are approximately met when training a very wide network with SP and SGD, where parameters in early layers move very little. This scenario is demonstrated in Fig. 6.1. In Fig. 6.10(a), we show that the features are largely unchanged after training, providing further evidence for this interpretation. This confirms that we are in the extended feature (EF) regime, where a small amount of additional feature alignment occurred beyond the baseline.

Thus, when the optimisation process remains confined to the vicinity of the initialisation, $P(f)$ serves as the primary inductive bias. If the initialisation lies within a region of poor inductive bias (e.g., the chaotic regime), $P(f)$ will not adequately favour simple functions, leading to subpar generalisation (Chapter 4). In this region, we have excellent answers for Q(i) and Q(ii).

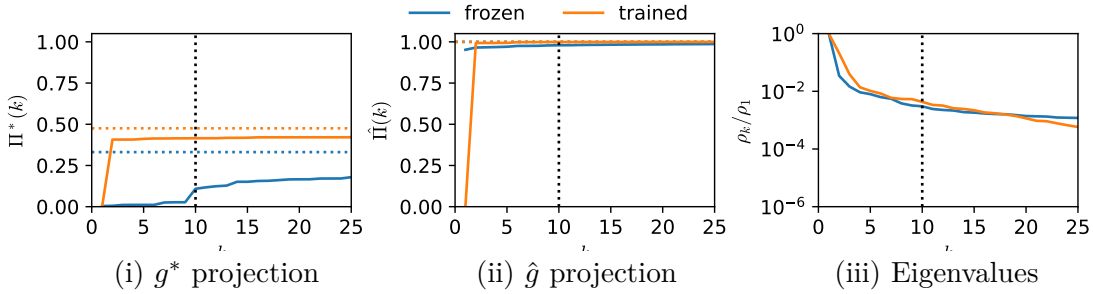
6.5.2 In the feature learning regime

We can determine whether feature learning has occurred by inspecting the network's effective kernel. A significant difference between the kernel at initialisation and the kernel corresponding to the final parameters indicates a substantial change in the features $\Phi_k(x)$. For example, the results in Fig. 6.2 imply that a CNN trained on CIFAR-10 must have undergone feature learning. Our method in Section 6.3 corroborates this, as shown in Fig. 6.10(b).

A second clear empirical indicator of a network's departure from its initial prior is the "kink" observed in the learning curve, depicted in Fig. 6.5. This kink signifies a transition from the NNGP-like EF regime to a rich feature learning regime, where the network actively discovers features better adapted to the data than those present at initialisation. What drives this transition?



(a) FCN on MNIST



(b) CNN on CIFAR-10

Figure 6.10: Feature learning for experiments in Section 6.1. (a) We use the same FCN on binarised MNIST (3-layer, width-1024, ReLU activations) as Fig. 6.1(a). (b) shows the CNN from Fig. 6.2(a). As predicted from those experiments, very little feature learning occurs for the FCN but a significant amount for the CNN – although the eigenvalue distributions barely change in both cases.

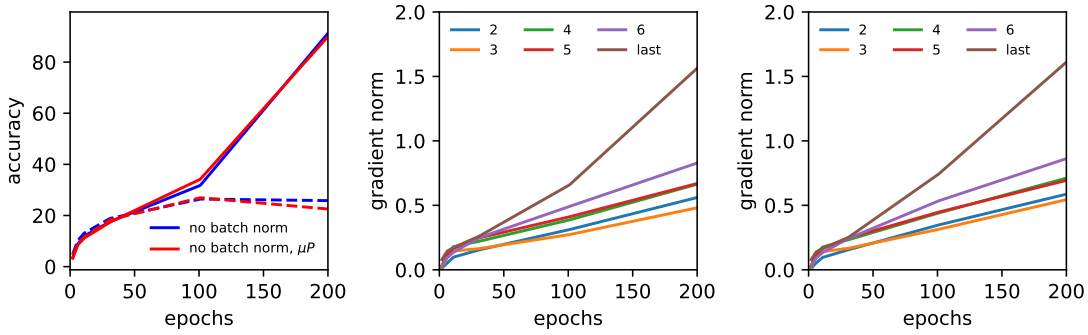
Width As shown by Seroussi et al. [140], architectural constraints like narrower widths compel a network towards feature learning. They demonstrate a significant shift in the network's kernel, indicating that narrower networks must learn data-dependent features to achieve good performance, thereby departing from their infinite-width NNGP limit – see Appendix B.3. Furthermore, when initial features are insufficient for data representation, feature learning is necessary for the model to fit the training data, as illustrated in Fig. 6.3.

Optimiser Hyperparameters The effect of width is not the only factor; optimiser hyperparameters also play a crucial role. Our findings in Fig. 6.6 unequivocally show that increasing the learning rate promotes greater feature learning. This can occur at any width, suggesting that treating the optimiser as a continuous process, as in [140], is not sufficient to understand how feature learning occurs.

This complexity stands in stark contrast to the toy model of Chapter 3. There, the result of feature learning was perfectly interpretable: with a single hyperparameter, weight decay, we could control the bias. Large weight decay caused the system to find the function representation with the minimum norm (corresponding to the minimum $K_{DNF}(f)$), while no weight decay led to sampling from $P(f)$ – providing a full answer to Q(ii). Furthermore, because the optimiser was fully Bayesian, we had an easy answer to Q(i).

The experiments in the previous section provided partial answers to Q(ii) – but did not offer a path to a clear theory of feature learning or answering Q(i).

6.5.3 Parameterisation and Feature Learning



(a) accuracy (b) no batch norm, SP (c) no batch norm, μP

Figure 6.11: Evaluating μP. As in Fig. 6.8, we show VGG16 on CIFAR-100, but compare SP to μP without batch norm. μP suffers from the same gradient problem as SP. This highlights one thing μP cannot fix: poor gradient propagation inherent to architectural depth. While μP correctly scales learning rates to balance the potential impact of each layer's updates, it cannot compensate for a gradient signal that is already attenuated. In deep sequential models like VGG without residual connections or normalisation, early layers receive a degraded signal, causing the final layers to dominate learning under both SP and μP.

What parameterisation changes (and what it cannot). Parameterisation controls both the infinite–width limit and the scale of layerwise updates. With Standard Parameterisation (SP) at infinite width and infinitesimal step size, training is in the lazy/NTK regime: the NTK is (approximately) constant and features remain close to initialization. NTK parameterisation enforces this limit by construction.

By contrast, μP preserves non-trivial feature learning at infinite width under proper learning-rate scaling. However, μP rescales updates; it does not create gradient signal where architecture destroys it. Fig. 6.11 compares VGG16 on CIFAR-100 without batch norm under SP versus μP. Despite correct μP scaling, early-layer gradients are still severely attenuated; the alignment peak sits late, and the final layers dominate. Measured kernel drift confirms the picture: Δ_Φ is small and Δ_{NTK} remains close to its initial value. In short, μP cannot compensate for architectural vanishing gradients (lack of BN/residual connections).

Lou Y, Mingard C, Hayou S. Feature learning and signal propagation in deep neural networks. In International Conference on Machine Learning 2022 Jun 28 (pp. 14248-14282). PMLR. [3]

Information loss predicts where features form. Lou et al. [3] study this problem of gradient attenuation. Deep sequential nets suffer attenuation of information both forward (activations) and backward (gradients). Theory based on signal propagation predicts a "peak-alignment" layer $\ell^\star$ that balances forward and backward loss; for depth L one expects $\ell^\star = \Theta(L^{3/5})$ up to a pre-factor that depends on optimiser hyperparameters [3]. Empirical observations suggest that larger learning rates reduce the constant factors, moving the peak alignment towards the start of the network. This correlates with better generalisation (cf. our layerwise $\Pi^*/\hat{\Pi}$ and gradient-energy profiles).

Göring N, London C, Erturk AH, Mingard C, Nam Y, Louis AA. Feature learning is decoupled from generalisation in high capacity neural networks. In High-dimensional Learning Dynamics, 2025. [10].

Feature change $\neq$ generalisation. Finally, stronger feature learning does not guarantee better test performance. In high-capacity settings we observe substantial Δ_Φ with little or no generalisation gains, consistent with Göring et al. [10]: the quality and placement of learned features matter more than their sheer magnitude. Within our MF/EF framework, the transition to the MF regime coincides with an

improved learning-curve exponent, but pushing alignment into very early layers (e.g. greedy layerwise training) can harm generalisation.

6.5.4 Towards a model for feature learning

Göring N, Mingard C, Nam Y, Louis AA. A simple mean field model of feature learning. Submitted to ICLR 2026. [12].

The rest of this chapter has been empirical. Here we sketch a simple model of feature learning from Göring et al. [12], and tie them to Q(i)/Q(ii).

From optimisation to a posterior. We start from the stationary distribution of stochastic gradient Langevin dynamics (SGLD) for a two-layer network with input dimension d, hidden dimension N, nonlinearity φ, and initialisation variances σ_a and σ_w:

$$f_\theta(x) = \frac{1}{N^\gamma}\sum_{i=1}^{N} a_i, \varphi(w_i^\top x), \quad a_i \sim \mathcal{N}(0, \sigma_a^2), \quad w_i \sim \mathcal{N}\Big(0, \tfrac{\sigma_w^2}{d}, I_d\Big). \tag{6.17}$$

The posterior of SGLD is given by Eq. (6.18) (see Appendix B.3 for more details).

$$p(\theta \mid S) \;\propto\; \exp\Big\{ -\frac{1}{2\kappa^2}\sum_{\mu=1}^{P}\Big(y_\mu - f_\theta(x_\mu)\Big)^2 \;-\; \frac{1}{2\sigma_a^2}\sum_{i=1}^{N} a_i^2 \;-\; \frac{d}{2\sigma_w^2}\sum_{i=1}^{N}\|w_i\|^2\Big\}. \tag{6.18}$$

The normalisation $N^{-\gamma}$ interpolates between NTK ($\gamma = \frac{1}{2}$) and mean–field ($\gamma = 1$).

The model. Directly analysing (6.18) is hard because the data term couples all neurons. We therefore use a *single–neuron mean–field ($\mathcal{MF}$)* approximation: replace global couplings by a self–consistent effective field, such that all neurons in the first layer are sampled i.i.d. distribution, that is a function of data and initialisation variance. How does this compare to familiar limits?

- **NNGP/NTK (no feature learning).** With $\gamma = \frac{1}{2}$ and $N \to \infty$, the kernel is fixed at initialization; features stay near their prior.
- **$\mathcal{MF}$ (feature learning).** Neuron weights are i.i.d. but from a *posterior–tilted* distribution determined by the data and initialisation, enabling feature movement.

$\mathcal{MF}$–ARD: allowing sparsification. Plain $\mathcal{MF}$ treats all coordinates symmetrically and underestimates the generalisation gains after the transition. Real nets sparsify: many neurons/coordinates shrink while a subset grows [6]. We capture this with $\mathcal{MF}$ plus *automatic relevance determination* (ARD) via per–coordinate precisions ρ_j:

$$p(w \mid \rho) = \prod_{j=1}^{d} \mathcal{N}!\Big(w_j \mid 0, \rho_j^{-1}\Big), \quad p(\rho) = \prod_{j=1}^{d} \mathrm{Gamma}\Big(\rho_j \mid \alpha_0, \beta_0\Big), \tag{6.19}$$

with $\mathbb{E}[\rho_j] = \alpha_0/\beta_0 = d/\sigma_w^2$. Marginalising ρ yields heavy–tailed $p(w_j)$: small coordinates are strongly shrunk while large, task–aligned coordinates escape, inducing sparsity and feature selection. This leads to a simple prediction for the onset of feature learning as a function of the effective noise/temperature. Writing k for task signal strength (e.g. teacher norm or margin scale), the critical variance scales as

$$\kappa_c^2 \asymp \Theta\Big(\tfrac{1}{\sqrt{d,k}}\Big) \quad (\text{plain } \mathcal{MF}), \qquad \kappa_c^2 \asymp \Theta\Big(\tfrac{1}{k}\Big) \quad (\mathcal{MF}\text{–ARD}). \tag{6.20}$$

At the transition, symmetry breaking induces an anisotropic low–rank deformation of the kernel: an outlier eigenvalue emerges along the task direction while the bulk stays almost unchanged. This symmetry breaking is likely of the same kind from Fig. 6.5 (note that κ_c is undefined for the $\gamma = \frac{1}{2}$, consistent with the figure).

Relation to Q(i)/Q(ii). This $\mathcal{MF}$ model assumes that we are training with SGLD – in the limit of infinitesimal learning rate, the optimiser is sampling exactly from a Bayesian posterior (in this approximation, Q(i) is answered: exactly). With respect to Q(ii), we can say the following. When feature learning occurs, the feature embeddings reorganise via a symmetry-breaking transition: an order parameter for task alignment becomes non-zero at a critical dataset size/noise level, and the learned kernel develops an outlier eigenvalue in the task direction. After this onset, a self-reinforcing *input feature selection* dynamic drives neuron- and coordinate-wise specialisation, yielding heavy-tailed weight marginals and sparsification so that only a few features dominate. In our $\mathcal{MF}$-ARD refinement, this appears as coordinate-dependent rescaling that nonlinearly deforms the kernel and boosts task-aligned

eigenmodes, producing an $\mathcal{O}(1)$ kernel change even at infinite width. Overall, feature learning manifests as an anisotropic, effectively low-rank deformation of the feature map that selectively amplifies task-relevant directions. This is likely to be the driver of the EF to MF transition observed earlier in this chapter.

7

Automatic Gradient Descent

This chapter introduces Automatic Gradient Descent (AGD), an optimisation framework that eliminates hyperparameter tuning by dynamically scaling updates using the layer's spectral norm. It derives AGD from first principles, showing that enforcing a spectral scaling condition yields stable, architecture-invariant learning dynamics that promote effective feature learning across network scales.

Bernstein, J., Mingard, C., Huang, K., Azizan, N. and Yue, Y., 2023. Automatic gradient descent: Deep learning without hyperparameters. arXiv preprint arXiv:2304.05187. [4]

In the preceding parts of this thesis, we explored generalisation by abstracting away the optimiser's specific role, examining fully Bayesian methods (Chapter 3) or systems where the optimiser's dynamics were approximately Bayesian (Chapter 4). After demonstrating the pivotal role an optimiser can play in shaping learned representations (Chapter 6), we now focus directly on the optimisation process itself.

First, however, we should establish the criteria for an effective optimiser.

Hyperparameter Stability An ideal optimiser should exhibit stability in its hyperparameters across varying model scales. For instance, significant changes to a network's width or depth should not necessitate a new, exhaustive search for a new set of optimum hyperparameters. Standard optimisers like SGD and

Adam often lack this property, as their optimal settings are highly sensitive to architectural details.

Minimal Hyperparameters The complexity of training should be minimised by reducing the number of tunable hyperparameters. Practitioners must often tune not only the learning rate but also its warmup and decay schedules, momentum, weight decay, and stopping criteria. These parameters have complex, non-obvious interactions, and even with established heuristics, finding an optimal configuration often devolves into costly trial and error. A desirable optimiser would reduce this manual burden. This would also include a prescribed way to normalise data and initialise weights.

Effective Feature Learning A fundamental goal of an optimiser is to facilitate feature learning across all model scales. It must prevent the training dynamics from collapsing into a 'lazy' regime, particularly in wide networks, ensuring that all layers actively contribute to learning new representations. While parameterisation schemes like μP were developed for this purpose, they are not without their own challenges [47], highlighting the need for an optimiser that inherently supports robust feature learning.

This chapter introduces Automatic Gradient Descent (AGD), a novel optimisation framework designed to meet these criteria. It uses a parameterisation designed to ensure stability and promote feature learning across diverse network architectures, eliminating the need for hyperparameter tuning (introduced in Section 7.1). The foundational principle of AGD is to achieve stable and effective feature learning by ensuring that weight updates are appropriately scaled relative to the weights themselves. AGD accomplishes this by enforcing a "spectral scaling condition" at each training step, dynamically adapting the update size to the local geometry of the loss landscape (introduced in Section 7.2). By deriving an optimiser from first principles, AGD provides a robust, parameter-free method for training deep neural networks.

7.1 Theory: Parameterisation

Automatic Gradient Descent parameterisation (AGD) was first introduced in [4] (which makes up this chapter), and was later expanded upon by Yang, Bernstein and Simon in [41]. It aims to achieve the balanced updates of feature-learning regimes like μP, automatically. The foundational principle is that for feature learning to occur, the spectral norm of weight matrices and their updates must scale appropriately – specifically like $\sqrt{d_{out}/d_{in}}$. AGD parameterisation implements this "spectral scaling condition" directly by dynamically re-scaling the weights and normalising the gradients by their spectral norm at each step. This approach contrasts with μP, which achieves the same scaling condition indirectly through carefully prescribed, width-dependent learning rates. The reason the spectral norm is the correct metric is that gradient updates are inherently low-rank – so normalising by the spectral norm correctly regulates the magnitude of these updates to ensure stable feature evolution. Yang et al. [41] shows that the early alignment problems μP has (see Section 2.6.2) do not apply for this parameterisation.

		Init. variance	Param. multiplier	Gradient ($\times\eta$)
μP	Input	1	1	n
	Inter	$n^{-1/2}$	1	1
	Readout	n^{-1}	1	n^{-1}
AGD	Input	$\sqrt{\frac{n}{d_{in}}}\frac{W^{(1)}}{\left\|W^{(1)}\right\|_2}$	1	$\sqrt{\frac{n}{d_{in}}}\frac{W^{(1)}}{\left\|W^{(1)}\right\|_2}$
	Inter	$\frac{W^{(l)}}{\left\|W^{(l)}\right\|_2}$	1	$\frac{W^{(l)}}{\left\|W^{(l)}\right\|_2}$
	Readout	$\sqrt{\frac{d_{out}}{n}}\frac{W^{(L)}}{\left\|W^{(L)}\right\|_2}$	1	$\sqrt{\frac{d_{out}}{n}}\frac{W^{(L)}}{\left\|W^{(L)}\right\|_2}$

Table 7.1: Parameterisations, variances of initialisation

AGD parameterisation on the toy model In the toy model from Section 2.6, the AGD parameterisation enforces weight matrix norms of $\|U\|_2^2 = n$ and $\|V\|_2^2 = n^{-1}$. This initial state is identical to that of μP, and consequently the network output $f = VUx$ also vanishes at initialization with a magnitude of $O(n^{-1/2})$. The

key difference lies in the update rule, which takes the form of normalized gradient descent with a dynamically adjusted step size.

Let's analyze the update to each layer under this scheme.

- The update for the first layer is $\Delta U = -(\eta n^{1/2})\frac{g_U}{\|g_U\|_2}$. Since the gradient $g_U \propto V^T$ has a norm $\|g_U\|_2 \propto \|V\|_2 = O(n^{-1/2})$, the update's norm is $\|\Delta U\|_2 = O(n^{1/2})$. This is the same order as the weight norm $\|U\|_2 = O(n^{1/2})$, ensuring active feature learning.

- The update for the second layer is $\Delta V = -(\eta n^{-1/2})\frac{g_V}{\|g_V\|_2}$. The gradient $g_V \propto U^T$ has a norm $\|g_V\|_2 \propto \|U\|_2 = O(n^{1/2})$, so the update's norm is $\|\Delta V\|_2 = O(n^{-1/2})$. This is also of the same order as the weight norm $\|V\|_2 = O(n^{-1/2})$, meaning this layer also trains non-trivially.

Now we analyse the update to the entire function. The change Δf is composed of two main terms, $V(\Delta U)$ and $(\Delta V)U$.

- The first term is $V(\Delta U) = -V(\eta n^{1/2})\frac{V^T}{\|V\|_2} = -\eta n^{1/2}\|V\|_2 = O(1)$.

- The second term is $(\Delta V)U = -(\eta n^{-1/2})\frac{U^T}{\|U\|_2}U = -\eta n^{-1/2}\|U\|_2 = O(1)$.

Just like in μP, both terms that drive the evolution of the function are stable and perfectly balanced. AGD can therefore be seen as an adaptive method that implicitly discovers the correct scaling factors required for feature learning, achieving the same desirable dynamics as μP without needing to pre-specify precise, width-dependent learning rates for each layer.

7.2 Theory: Automatic Learning Rate

Given a vector $\boldsymbol{v}$ in $\mathbb{R}^n$, we will need to measure its size in three different ways:

Definition 14 (Manhattan norm)**.** *The Manhattan norm $\|\cdot\|_1$ of a vector $\boldsymbol{v}$ is defined by $\|\boldsymbol{v}\|_1 := \sum_i |\boldsymbol{v}_i|$.*

Definition 15 (Euclidean norm)**.** *The Euclidean norm $\|\cdot\|_2$ of a vector $\boldsymbol{v}$ is defined by $\|\boldsymbol{v}\|_2 := \sqrt{\sum_i \boldsymbol{v}_i^2}$.*

Definition 16 (Infinity norm)**.** *The infinity norm* $\|\cdot\|_\infty$ *of a vector* $\boldsymbol{v}$ *is defined by* $\|\boldsymbol{v}\|_\infty := \max_i |\boldsymbol{v}_i|$.

For a matrix $\boldsymbol{M}$ in $\mathbb{R}^{m\times n}$, the reader should be aware that it has a singular value decomposition:

Fact 1 (SVD)**.** *Every matrix* $\boldsymbol{M}$ *in* $\mathbb{R}^{m\times n}$ *admits a singular value decomposition (SVD) of the form* $\boldsymbol{M} = \sum_{i=1}^{\min(m,n)} \sigma_i(\boldsymbol{M}) \cdot \boldsymbol{u}_i \boldsymbol{v}_i^\top$ *where the left singular vectors* $\{\boldsymbol{u}_i\}$ *are orthonormal vectors in* $\mathbb{R}^m$*, the right singular vectors* $\{\boldsymbol{v}_i\}$ *are orthonormal vectors in* $\mathbb{R}^m$ *and the singular values* $\{\sigma_i(\boldsymbol{M})\}$ *are non-negative scalars.*

The singular value decomposition allows us to measure the size of a matrix in two different ways:

Definition 17 (Frobenius norm)**.** *The Frobenius norm* $\|\cdot\|_F$ *of a matrix* $\boldsymbol{M}$ *is given by* $\|\boldsymbol{M}\|_F := \sqrt{\sum_i \sigma_i(\boldsymbol{M})^2}$.

Definition 18 (Operator norm)**.** *The operator norm* $\|\cdot\|_*$ *of a matrix* $\boldsymbol{M}$ *is given by* $\|\boldsymbol{M}\|_* := \max_i \sigma_i(\boldsymbol{M})$.

While the operator norm $\|\boldsymbol{M}\|_*$ reports the largest singular value, the quantity $\|\boldsymbol{M}\|_F / \sqrt{\min(m,n)}$ reports the root mean square singular value. Finally, we will need to understand two aspects of matrix conditioning:

Definition 19 (Rank)**.** *The rank of a matrix counts the number of non-zero singular values.*

Definition 20 (Stable rank)**.** *The stable rank of a matrix* $\boldsymbol{M}$ *is defined by* $\operatorname{rank}_{\text{stable}} \boldsymbol{M} := \|\boldsymbol{M}\|_F^2 / \|\boldsymbol{M}\|_*^2$.

The stable rank provides an approximation to the rank that ignores the presence of very small singular values. Let us consider the extremes. An orthogonal matrix $\boldsymbol{O} \in \mathbb{R}^{m\times n}$ has both full rank and full stable rank: $\operatorname{rank} \boldsymbol{O} = \operatorname{rank}_{\text{stable}} \boldsymbol{O} = \min(m,n)$. A rank-one matrix $\boldsymbol{P}$ has unit stable rank and satisfies $\|\boldsymbol{P}\|_* = \|\boldsymbol{P}\|_F$.

7.2.1 Majorise-Minimise for Generic Learning Problems

This section develops a framework for applying the majorise-minimise meta-algorithm to generic optimisation problems in machine learning. In particular, the novel technique of *functional expansion* is introduced. Section 7.2.2 will specialise this technique to deep neural networks. All proofs are supplied in Appendix H.1.

Given a machine learning model and a set of training data, our objective is to minimise the error of the model, averaged over the training data. Formally, we would like to minimise the following function:

Definition 21 (Composite objective)**.** *Consider a machine learning model $\boldsymbol{f}$ that maps an input $\boldsymbol{x}$ and a weight vector $\boldsymbol{w}$ to output $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w})$. Given data S and a convex loss function ℓ, the objective $\mathcal{L}(\boldsymbol{w})$ is defined by:*

$$\mathcal{L}(\boldsymbol{w}) := \frac{1}{|\mathsf{S}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \ell(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y}).$$

We refer to this objective as *composite* since the loss function ℓ is *composed* with a machine learning model $\boldsymbol{f}$. While the loss function itself is convex, the overall composite is often non-convex due to the non-linear machine learning model. Common convex loss functions include the square loss and the cross-entropy loss:

Example 7.2.1 (Square loss)**.** *The square loss is defined by:* $\ell(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y}) := \frac{1}{2d_L}\|\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w}) - \boldsymbol{y}\|_2^2$.

Example 7.2.2 (Xent loss)**.** *The cross-entropy (xent) loss is defined by:* $\ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) := -\log[\text{softmax}(\boldsymbol{f}(\boldsymbol{x}))]^\top \boldsymbol{y}$*, where the softmax function is defined by* $\text{softmax}(\boldsymbol{f}(\boldsymbol{x})) := \exp \boldsymbol{f}(\boldsymbol{x})/\|\exp \boldsymbol{f}(\boldsymbol{x})\|_1$.

Decomposition of linearisation error

First-order optimisers leverage the linearisation of the objective at the current iterate. To design such methods, we must understand the realm of validity of this linearisation. To that end, we derive a general decomposition of the linearisation error of a machine learning system. The result is stated in terms of a *perturbation hierarchy*. In particular, perturbing the weight vector of a machine learning model

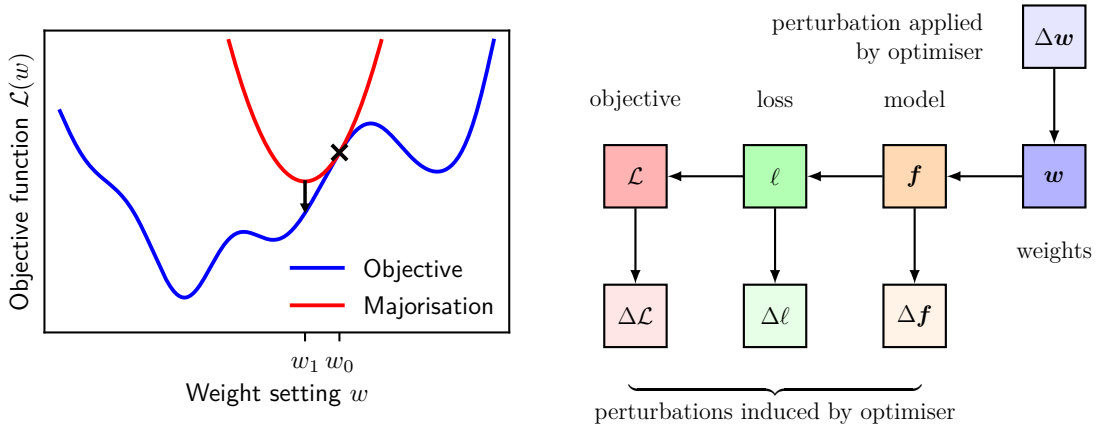


Figure 7.1: Majorise-minimise and the perturbation hierarchy. The **left panel** depicts the majorise-minimise meta-algorithm [141], which is an algorithmic pattern for reducing an objective (blue) by minimising a sequence of upper bounds (one shown in red). The upper bounds, known as a *majorisation*, must lie tangent to the objective to guarantee an improvement in one step of the meta-algorithm. The **right panel** depicts the perturbation hierarchy of a generic machine learning model: the optimiser perturbs the weights and this induces perturbations to the model output, the loss on individual training examples and ultimately the overall objective. Majorising machine learning objective functions requires addressing the full perturbation hierarchy.

$\boldsymbol{w} \to \boldsymbol{w} + \Delta \boldsymbol{w}$ induces perturbations to the model output $\boldsymbol{f} \to \boldsymbol{f} + \Delta \boldsymbol{f}$, to the loss on individual data samples $\ell \to \ell + \Delta \ell$ and, at last, to the overall objective function $\mathcal{L} \to \mathcal{L} + \Delta \mathcal{L}$. Formally, a weight perturbation $\Delta \boldsymbol{w}$ induces:

$$\Delta \boldsymbol{f}(\boldsymbol{x}) \quad \coloneqq \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w} + \Delta \boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w}); \qquad \text{(functional perturbation)}$$

$$\Delta \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) \quad \coloneqq \ell(\boldsymbol{f}(\boldsymbol{x}) + \Delta \boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) - \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}); \qquad \text{(loss perturbation)}$$

$$\Delta \mathcal{L}(\boldsymbol{w}) \quad \coloneqq \tfrac{1}{|\mathsf{S}|} \textstyle\sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathsf{S}} \Delta \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}). \qquad \text{(objective perturbation)}$$

We have adopted a compact notation where the dependence of $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w})$ on $\boldsymbol{w}$ is at times suppressed. The perturbation hierarchies of a generic machine learning model and a deep neural network are visualised in Figs. 7.1 and 7.2, respectively. The linearisation error of the objective perturbation $\Delta \mathcal{L}$ decomposes as:

Proposition 7.2.1 (Decomposition of linearisation error)**.** *For any differentiable loss ℓ and any differentiable machine learning model $\boldsymbol{f}$ the linearisation error of the*

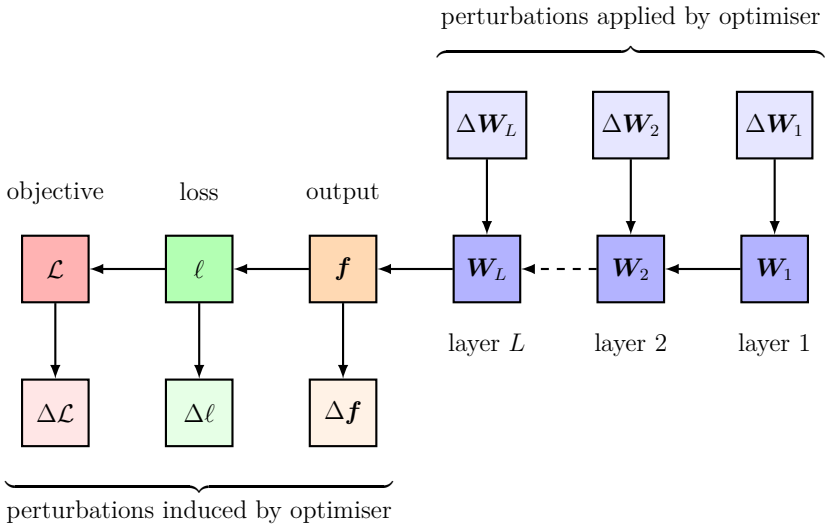


Figure 7.2: Perturbation hierarchy of a deep neural network. When training a neural network, the optimiser applies structured perturbations to the weights, in the form of one perturbation matrix $\Delta \boldsymbol{W}_k$ per weight matrix $\boldsymbol{W}_k$. Deep relative trust [142] provides a tool to understand how structured weight perturbations of this form affect the network output $\boldsymbol{f}$. Combining deep relative trust with a Bregman divergence [143] allows us to analyse the full perturbation hierarchy.

objective function $\mathcal{L}$ admits the following decomposition:

$$\underbrace{\Delta \mathcal{L}(\boldsymbol{w}) - \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w})^\top \Delta \boldsymbol{w}}_{\textit{linearisation error of objective}} = \frac{1}{|\mathsf{S}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathsf{S}} \nabla_{\boldsymbol{f}(\boldsymbol{x})} \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y})^\top \underbrace{[\Delta \boldsymbol{f}(\boldsymbol{x}) - \nabla_{\boldsymbol{w}} \boldsymbol{f}(\boldsymbol{x}) \Delta \boldsymbol{w}]}_{\textit{linearisation error of model}}$$
$$+ \frac{1}{|\mathsf{S}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathsf{S}} \underbrace{\Delta \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) - \nabla_{\boldsymbol{f}(\boldsymbol{x})} \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y})^\top \Delta \boldsymbol{f}(\boldsymbol{x})}_{\textit{linearisation error of loss}}.$$

In words: the linearisation error of the objective decomposes into two terms. The first term depends on the linearisation error of the machine learning model and the second term the linearisation error of the loss. This decomposition relies on nothing but differentiability. Proposition 7.2.1 may also be seen as a generalisation of the Gauss-Newton decomposition of the Hessian that holds to all orders. For a convex loss, the second term may be interpreted as a Bregman divergence:

Definition 22 (Bregman divergence of loss)**.** *For any convex loss ℓ:*

$$\text{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{f}(\boldsymbol{x}), \Delta \boldsymbol{f}(\boldsymbol{x})) := \Delta \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) - \nabla_{\boldsymbol{f}(\boldsymbol{x})} \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y})^\top \Delta \boldsymbol{f}(\boldsymbol{x}).$$

A Bregman divergence is just the linearisation error of a convex function. Two important examples are:

Lemma 1 (Bregman divergence of square loss)**.** *When ℓ is set to square loss, then:*

$$\operatorname{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{f}(\boldsymbol{x}), \Delta\boldsymbol{f}(\boldsymbol{x})) = \tfrac{1}{2d_L}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_2^2.$$

Lemma 2 (Bregman divergence of xent loss)**.** *When ℓ is set to cross-entropy loss, and if $\boldsymbol{y}^\top\mathbf{1} = 1$, then:*

$$\begin{aligned}\operatorname{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{f}(\boldsymbol{x}), \Delta\boldsymbol{f}(\boldsymbol{x})) &= D_{\mathrm{KL}}\Big(\operatorname{softmax}(\boldsymbol{f}(\boldsymbol{x}))\,\Big\|\,\operatorname{softmax}(\boldsymbol{f}(\boldsymbol{x}) + \Delta\boldsymbol{f}(\boldsymbol{x}))\Big)\\ &\leq \tfrac{1}{2}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_\infty^2 + \mathcal{O}(\Delta\boldsymbol{f}^3).\end{aligned}$$

Our methods may be applied to other convex losses by calculating or bounding their Bregman divergence.

Functional expansion and functional majorisation

Before continuing, we make one simplifying assumption. Observe that the first term on the right-hand side of Proposition 7.2.1 is a high-dimensional inner product between two vectors. Since there is no clear reason why these two vectors should be aligned, let us assume for convenience that their inner product is zero:

Assumption 1 (Orthogonality of model linearisation error)**.** *In the same setting as Proposition 7.2.1:*

$$\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top \underbrace{[\Delta\boldsymbol{f}(\boldsymbol{x}) - \nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x})\Delta\boldsymbol{w}]}_{\textit{linearisation error of model}} = 0.$$

While one can work without this assumption in special cases [144], we found that its inclusion simplifies the analysis and in practice did not lead to a discernible weakening of the resulting algorithm. In any case, this assumption is considerably milder than the common assumption in the literature [145, 146] that the model linearisation error is itself zero: $[\Delta\boldsymbol{f}(\boldsymbol{x}) - \nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x})\Delta\boldsymbol{w}] = 0$.

Armed with Proposition 7.2.1 and Assumption 1, we are ready to introduce functional expansion and majorisation:

Theorem 4 (Functional expansion)**.** *Consider a convex differentiable loss ℓ and a differentiable machine learning model $\boldsymbol{f}$. Under Assumption 1, the corresponding composite objective $\mathcal{L}$ admits the expansion:*

$$\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) = \underbrace{\mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top \Delta\boldsymbol{w}}_{\textit{first-order Taylor series}} + \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \operatorname{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{f}(\boldsymbol{x}), \Delta\boldsymbol{f}(\boldsymbol{x})).$$

So the perturbed objective $\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w})$ may be written as the sum of its first-order Taylor expansion with a Bregman divergence in the model outputs averaged over the training set. It is straightforward to specialise this result to different losses by substituting in their Bregman divergence:

Corollary 7.2.1 (Functional expansion of mean squared error)**.** *Under Assumption 1, for square loss:*

$$\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top \Delta\boldsymbol{w} + \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \tfrac{1}{2d_L}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_2^2.$$

Corollary 7.2.2 (Functional majorisation for xent loss)**.** *Under Assumption 1, for cross-entropy loss, if $\boldsymbol{y}^\top\mathbf{1} = 1$:*

$$\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) \leq \mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top \Delta\boldsymbol{w} + \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \tfrac{1}{2}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_\infty^2 + \mathcal{O}(\Delta\boldsymbol{f}^3).$$

When the functional perturbation is reasonably "spread out", we would expect $\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_\infty^2 \approx \|\Delta\boldsymbol{f}(\boldsymbol{x})\|_2^2/d_L$. In this setting, the functional majorisation of cross-entropy loss agrees with the functional expansion of mean squared error to second order. While the paper derives automatic gradient descent for the square loss, this observation justifies its application to cross-entropy loss, as in the case of the ImageNet experiments.

Recovering existing frameworks

We briefly observe that three existing optimisation frameworks may be recovered efficiently from Theorem 4:

Mirror descent For linear models $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{W}) := \boldsymbol{W}\boldsymbol{x}$, the Bregman divergence $\text{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{f}(\boldsymbol{x}), \Delta\boldsymbol{f}(\boldsymbol{x}))$ may be written $\text{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{W}\boldsymbol{x}, \Delta\boldsymbol{W}\boldsymbol{x})$. This is a convex function of the weight perturbation $\Delta\boldsymbol{W}$. Substituting into Theorem 4 and minimising with respect to $\Delta\boldsymbol{W}$ is the starting point for mirror descent.

Gauss-Newton method Substituting the linearised functional perturbation $\Delta\boldsymbol{f}(\boldsymbol{x}) \approx \nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x})\Delta\boldsymbol{w}$ into Corollary 7.2.1 and minimising with respect to $\Delta\boldsymbol{w}$ is the starting point for the Gauss-Newton method.

Natural gradient descent Substituting the linearised functional perturbation $\Delta\boldsymbol{f}(\boldsymbol{x}) \approx \nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x})\Delta\boldsymbol{w}$ into Corollary 7.2.2 and minimising with respect to $\Delta\boldsymbol{w}$ is the starting point for natural gradient descent.

7.2.2 Majorise-Minimise for Deep Learning Problems

In this section, we will focus our efforts on deriving an optimiser for deep fully-connected networks trained with square loss. The derivation for cross-entropy loss is analogous. Proofs are relegated to Appendix H.1.

Definition 23 (Fully-connected network)**.** *A fully-connected network (FCN)* $\boldsymbol{f}$ *of depth* L *maps an input* $\boldsymbol{x} \in \mathbb{R}^{d_0}$ *to an output* $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w}) \in \mathbb{R}^{d_L}$ *via* L *matrix multiplications interspersed by non-linearity* $\text{relu}(z) := \max(0, z)$*:*

$$\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w}) \coloneqq \boldsymbol{W}_L \circ (\text{relu} \circ \boldsymbol{W}_{L-1}) \circ (\text{relu} \circ \boldsymbol{W}_{L-2}) \circ \cdots \circ (\text{relu} \circ \boldsymbol{W}_1\boldsymbol{x}).$$

In this expression, $\boldsymbol{w}$ denotes the tuple of matrices $\boldsymbol{w} = (\boldsymbol{W}_1, ..., \boldsymbol{W}_L)$ with kth matrix $\boldsymbol{W}_k$ in $\mathbb{R}^{d_k \times d_{k-1}}$. In what follows, we will find the following dimensional scaling to be particularly convenient:

Prescription 1 (Dimensional scaling)**.** *For $\eta > 0$, the data $(\boldsymbol{x}, \boldsymbol{y})$, weights $\boldsymbol{W}_k$ and updates $\Delta \boldsymbol{W}_k$ should obey:*

$$\begin{aligned}
\|\boldsymbol{x}\|_2 &= \sqrt{d_0}; && && \text{(input scaling)}\\
\|\boldsymbol{W}_k\|_* &= \sqrt{d_k/d_{k-1}} && \text{for all } k = 1, ..., L; && \text{(weight scaling)}\\
\|\Delta \boldsymbol{W}_k\|_* &= \sqrt{d_k/d_{k-1}} \cdot \tfrac{\eta}{L} && \text{for all } k = 1, ..., L; && \text{(update scaling)}\\
\|\boldsymbol{y}\|_2 &= \sqrt{d_L}. && && \text{(target scaling)}
\end{aligned}$$

This is the AGD parameterisation from Section 7.1. While results can be derived without adopting Prescription 1, the scalings substantially simplify our formulae. One reason for this is that, under Prescription 1, we have the telescoping property that $\prod_{k=1}^{L} \|\boldsymbol{W}_k\|_* = \sqrt{d_L/d_0}$. For a concrete example of how this helps, consider the following bound on the norm of the network outputs:

Lemma 3 (Output bound)**.** *The output norm of a fully-connected network $\boldsymbol{f}$ obeys the following bound:*

$$\|\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w})\|_2 \leq \left[\prod_{k=1}^{L} \|\boldsymbol{W}_k\|_*\right] \times \|\boldsymbol{x}\|_2 = \sqrt{d_L} \text{ under Prescription 1.}$$

So, under Prescription 1, the bound is simple. Furthermore, the scaling of the update with a single parameter η reduces the problem of solving for an optimiser to a single parameter problem. To see how this might make life easier, consider the following lemma that relates weight perturbations to functional perturbations:

Lemma 4 (Deep relative trust)**.** *When adjusting the weights $\boldsymbol{w} = (\boldsymbol{W}_1, ..., \boldsymbol{W}_L)$ of a fully-connected network $\boldsymbol{f}$ by $\Delta \boldsymbol{w} = (\Delta \boldsymbol{W}_1, ..., \Delta \boldsymbol{W}_L)$, the induced functional perturbation $\Delta \boldsymbol{f}(\boldsymbol{x}) := \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w} + \Delta \boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w})$ obeys:*

$$\begin{aligned}
\|\Delta \boldsymbol{f}(\boldsymbol{x})\|_2 &\leq \left[\prod_{k=1}^{L} \|\boldsymbol{W}_k\|_*\right] \times \|\boldsymbol{x}\|_2 \times \left[\prod_{k=1}^{L} \left(1 + \frac{\|\Delta \boldsymbol{W}_k\|_*}{\|\boldsymbol{W}_k\|_*}\right) - 1\right]\\
&\leq \sqrt{d_L} \times (\exp \eta - 1) \text{ under Prescription 1.}
\end{aligned}$$

So, under Prescription 1, the single parameter η directly controls the size of functional perturbations.

In terms of enforcing Prescription 1 in practice, the norms of the data $(\boldsymbol{x}, \boldsymbol{y})$ may be set via pre-processing, the norm of the update $\Delta \boldsymbol{W}_k$ may be set via the optimisation algorithm and the norm of the weight matrix $\boldsymbol{W}_k$ may be set by the choice of initialisation. While, yes, $\|\boldsymbol{W}_k\|_*$ may drift during training, the amount that this can happen is limited by Weyl [147]'s inequality for singular values. In particular, after one step the perturbed operator norm $\|\boldsymbol{W}_k + \Delta \boldsymbol{W}_K\|_*$ is sandwiched like $(1 - \eta/L) \cdot \|\boldsymbol{W}_k\|_* \leq \|\boldsymbol{W}_k + \Delta \boldsymbol{W}_K\|_* \leq (1 + \eta/L) \cdot \|\boldsymbol{W}_k\|_*$. While this drift may accumulate over steps, some form of projection could be used to correct for this, but we defer the empirical study of this kind of projection to future work.

Deriving automatic gradient descent

With both functional majorisation and deep relative trust in hand, we can majorise the deep network objective:

Lemma 5 (Exponential majorisation)**.** *For an FCN with square loss, under Assumption 1 and Prescription 1:*

$$\mathcal{L}(\boldsymbol{w} + \Delta \boldsymbol{w}) \leq \mathcal{L}(\boldsymbol{w}) + \frac{\eta}{L} \sum_{k=1}^{L} \left[\sqrt{d_k / d_{k-1}} \times \operatorname{tr} \frac{\Delta \boldsymbol{W}_k^\top \nabla_{\boldsymbol{W}_k} \mathcal{L}}{\|\Delta \boldsymbol{W}_k\|_*} \right] + \tfrac{1}{2} (\exp \eta - 1)^2.$$

Observe that the majorisation only depends on the magnitude of the scalar η and on some notion of alignment $\operatorname{tr} \Delta \boldsymbol{W}_k^\top \nabla_{\boldsymbol{W}_k} \mathcal{L} / \|\Delta \boldsymbol{W}_k\|_*$ between the perturbation matrix $\Delta \boldsymbol{W}_k$ and the gradient matrix $\nabla_{\boldsymbol{W}_k} \mathcal{L}$. To derive an optimiser, we would now like to minimise this majorisation with respect to η and this angle. First, let us introduce one additional assumption and one additional definition:

Assumption 2 (Gradient conditioning)**.** *The gradient satisfies* $\operatorname{rank}_{\text{stable}} \nabla_{\boldsymbol{W}_k} \mathcal{L} = 1$ *at all layers* $k = 1, ..., L$.

This assumption implies that the Frobenius norm $\|\nabla_{\boldsymbol{W}_k} \mathcal{L}\|_F$ and operator norm $\|\nabla_{\boldsymbol{W}_k} \mathcal{L}\|_*$ of the gradient at layer k are equal. It is not immediately obvious why this should be a good assumption. After all, the gradient is a sum of $|\mathsf{S}|$ rank-one matrices: $\nabla_{\boldsymbol{W}_k} \mathcal{L} = \frac{1}{|\mathsf{S}|} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathsf{S}} \nabla_{\boldsymbol{h}_k} \ell(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y}) \otimes \boldsymbol{h}_{k-1}$, where $\boldsymbol{h}_{k-1}(\boldsymbol{x})$ and $\boldsymbol{h}_k(\boldsymbol{x})$ denote the inputs and outputs of the weight matrix $\boldsymbol{W}_k$ at layer k, and $\otimes$ denotes

the outer product. So, naïvely, one might expect the gradient $\nabla_{\boldsymbol{W}_k}\mathcal{L}$ to have a stable rank of $\min(d_k, d_{k-1}, |\mathsf{S}|)$. But it turns out to be a good assumption in practice [42, 46, 41]. And for the definition:

Definition 24 (Gradient summary)**.** *At a weight setting $\boldsymbol{w}$, the gradient summary G is given by:*

$$G := \frac{1}{L}\sum_{k=1}^{L}\sqrt{d_k/d_{k-1}}\cdot\|\nabla_{\boldsymbol{W}_k}\mathcal{L}(\boldsymbol{w})\|_F.$$

The gradient summary is a weighted average of gradient norms over layers. It can be thought of as a way to measure the size of the gradient while accounting for the fact that the weight matrices at different layers may be on different scales. This is related to the concept of the *gradient scale coefficient* of Philipp et al. [148].

We now have everything we need to derive automatic gradient descent via the majorise-minimise principle:

Theorem 5 (Automatic gradient descent)**.** *For a deep fully-connected network, under Assumptions 1 and 2 and Prescription 1, the majorisation of square loss given in Lemma 5 is minimised by setting:*

$$\eta = \log\frac{1+\sqrt{1+4G}}{2}, \qquad \Delta\boldsymbol{W}_k = -\frac{\eta}{L}\cdot\sqrt{d_k/d_{k-1}}\cdot\frac{\nabla_{\boldsymbol{W}_k}\mathcal{L}}{\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F}, \qquad \textit{for all layers } k = 1, ..., L.$$

We present pseudocode for this theorem in Algorithm 1, and a PyTorch implementation in Appendix H.2. Via a simple derivation based on clear algorithmic principles, automatic gradient descent unifies various heuristic and theoretical ideas that have appeared in the literature:

- *Relative updates.* The update is scaled relative to the norm of the weight matrix to which it is applied—assuming the weight matrices are scaled according to Prescription 1. Such a scaling was proposed by You et al. [149] and further explored by Carbonnelle and Vleeschouwer [150] and Bernstein et al. [142]. There is evidence that such relative synaptic updates may occur in neuroscience [151].

Algorithm 1 Automatic gradient descent. The matrix $\boldsymbol{W}_k$ in $\mathbb{R}^{d_k \times d_{k-1}}$ is the weight matrix at layer k. The gradient $\nabla_{\boldsymbol{W}_k}\mathcal{L}$ is with respect to the objective $\mathcal{L}$ evaluated on a mini-batch B of training samples.

Function initialise_weights()
for layer k in $\{1, ..., L\}$: **do**
  $\boldsymbol{W}_k \sim \text{uniform}(\text{orthogonal}(d_k, d_{k-1}))$ {sample semi-orthogonal matrix}
  $\boldsymbol{W}_k \leftarrow \boldsymbol{W}_k \cdot \sqrt{\frac{d_k}{d_{k-1}}}$ {rescale its singular values}
end for

Function update_weights()
$G \leftarrow \frac{1}{L}\sum_{l=1}^{L} \|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F \cdot \sqrt{\frac{d_k}{d_{k-1}}}$ {Compute gradient summary}
$\eta \leftarrow \log \frac{1+\sqrt{1+4G}}{2}$ {set automatic learning rate}
for layer k in $\{1, ..., L\}$: **do**
  $\boldsymbol{W}_k \leftarrow \boldsymbol{W}_k - \frac{\eta}{L} \cdot \frac{\nabla_{\boldsymbol{W}_k}\mathcal{L}}{\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F} \cdot \sqrt{\frac{d_k}{d_{k-1}}}$ {update weights}
end for

- *Depth scaling.* Scaling the perturbation strength like $1/L$ for networks of depth L was proposed on theoretical grounds by Bernstein et al. [142] based on analysis via deep relative trust.

- *Width scaling.* The dimensional factors of d_k and d_{k-1} that appear closely relate to the maximal update parameterisation of Yang and Hu [40] designed to ensure hyperparameter transfer across network width. This connection is further explicated by Yang et al. [41].

- *Gradient clipping.* The logarithmic dependence of the update on the gradient summary may be seen as an automatic form of *adaptive gradient clipping* [152]—a technique which clips the gradient once its magnitude surpasses a certain threshold set by a hyperparameter.

Convergence analysis

This section presents theoretical convergence rates for automatic gradient descent. While the spirit of the analysis is standard in optimisation theory, the details may still prove interesting for their detailed characterisation of the optimisation properties of deep networks. For instance, we propose a novel Polyak-Łojasiewicz inequality tailored to the operator structure of deep networks. We begin with two observations:

Lemma 6 (Bounded objective). *For square loss, the objective is bounded as follows:*

$$\mathcal{L}(\boldsymbol{w}) \leq \frac{1}{|\mathsf{S}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \frac{\|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})\|_2^2 + \|\boldsymbol{y}\|_2^2}{2d_L} \leq 1 \text{ under Prescription 1.}$$

Lemma 7 (Bounded gradient). *For square loss, the norm of the gradient at layer k is bounded as follows:*

$$\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F \leq \frac{\prod_{l=1}^{L}\|\boldsymbol{W}_l\|_*}{\|\boldsymbol{W}_k\|_*} \cdot \sqrt{\frac{2\mathcal{L}(\boldsymbol{w})}{d_L}} \cdot \sqrt{\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\|\boldsymbol{x}\|_2^2} \leq \sqrt{2\cdot\frac{d_{k-1}}{d_k}}$$

under Prescription 1.

These results help us prove that automatic gradient descent converges to a point where the gradient vanishes:

Lemma 8 (Convergence rate to critical point). *Consider a fully-connected network trained by automatic gradient descent (Theorem 5) and square loss for T iterations. Let G_t denote the gradient summary (Definition 24) at step $t \leq T$. Under Assumptions 1 and 2 and supposing that Prescription 1 is maintained throughout training, AGD converges at the following rate:*

$$\min_{t\in\{1,\ldots,T\}} G_t^2 \leq \frac{11}{T}.$$

This lemma can be converted into a convergence rate to a global minimum with one additional assumption:

Assumption 3 (Deep Polyak-Łojasiewicz inequality). *For some $\alpha > 0$, the gradient norm is lower bounded by:*

$$\begin{aligned}\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F &\geq \alpha \times \frac{\prod_{l=1}^{L}\|\boldsymbol{W}_l\|_*}{\|\boldsymbol{W}_k\|_*} \cdot \sqrt{\frac{2\mathcal{L}(\boldsymbol{w})}{d_L}} \cdot \sqrt{\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\|\boldsymbol{x}\|_2^2} \\ &= \alpha \times \sqrt{2\cdot\mathcal{L}(\boldsymbol{w})\cdot\frac{d_{k-1}}{d_k}} \text{ under Prescription 1.}\end{aligned}$$

This lower bound mirrors the structure of the upper bound in Lemma 7. The parameter α captures how much of the gradient is attenuated by small singular values in the weights and by deactivated relu units. While Polyak-Łojasiewicz inequalities are common in the literature [153], our assumption is novel in that it pays attention to the operator structure of the network. Assumption 3 leads to the following theorem:

Theorem 6 (Convergence rate to global minima)**.** *For automatic gradient descent (Theorem 5) in the same setting as Lemma 8 but with the addition of Assumption 3, the mean squared error objective at step T obeys:*

$$\mathcal{L}(\boldsymbol{w}_T) \leq \frac{1}{\alpha^2} \times \frac{6}{T}.$$

7.3 Results

The goal of our experiments was twofold. First, we wanted to test automatic gradient descent (AGD, Algorithm 1) on a broad variety of network architectures and datasets to check that it actually works. In particular, we tested AGD on fully-connected networks (FCNs, Definition 23), and both VGG-style [117] and ResNet-style [118] convolutional neural networks on the CIFAR-10, CIFAR-100 [154] and ImageNet [155, ILSVRC2012] datasets with standard data augmentation. And second, to see what AGD may have to offer beyond the status quo, we wanted to compare AGD to tuned Adam and SGD baselines, as well as Adam and SGD run with their default hyperparameters.

A general theme from the experiments was that whilst AGD can train all architectures, its performance sometimes falls below that of a fully tuned baseline, both in terms of generalization and rate of convergence. While it would be nice if AGD matched the performance of fully tuned optimizers, it performed well enough to suggest that hyperparameter-free training is possible with our framework.

To get AGD working with convolutional layers, we adopted a per-submatrix normalisation scheme. Specifically, for a convolutional tensor with filters of size $\texttt{k}_\texttt{x} \times \texttt{k}_\texttt{y}$, we implemented the normalisation separately for each of the $\texttt{k}_\texttt{x} \times \texttt{k}_\texttt{y}$ submatrices of dimension $\texttt{channels}_\texttt{in} \times \texttt{channels}_\texttt{out}$. Since AGD does not yet support biases or affine parameters in batchnorm, we disabled these parameters in all architectures. To at least adhere to Prescription 1 at initialisation, AGD draws initial weight matrices uniform semi-orthogonal and re-scaled by a factor of $\sqrt{\texttt{fan\_in}/\texttt{fan\_out}}$. Adam and SGD baselines used the PyTorch default initialisation. A PyTorch implementation of AGD reflecting these details is given in Appendix H.2. All

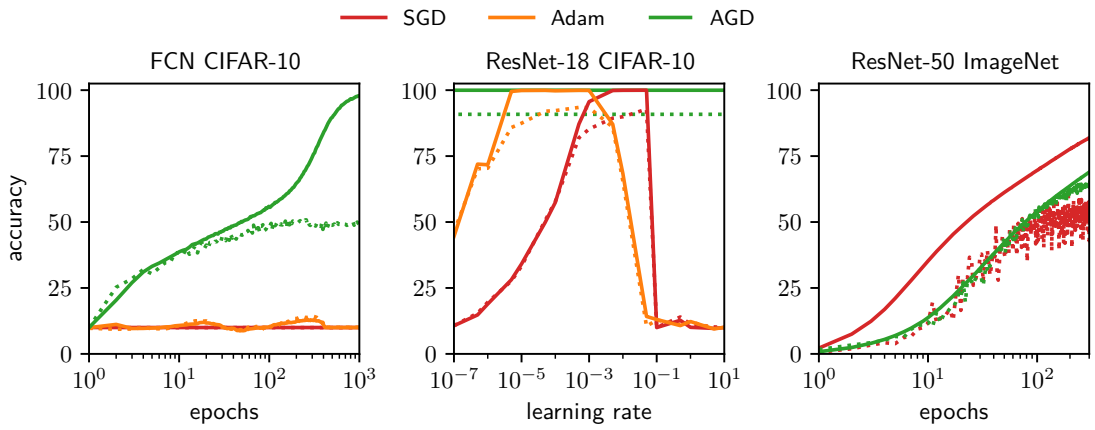


Figure 7.3: Automatic gradient descent trains neural networks reliably without hyperparameters. Solid lines show training accuracy and dotted lines show test accuracy. The networks are unregularised with biases and affine parameters disabled, as these features are not yet supported by AGD. In the **left panel**—unlike AGD—Adam and SGD failed to train a 32-layer fully-connected network on CIFAR-10 with their default learning rates of 0.001 for Adam and 0.1 for SGD. The **middle panel** displays a learning rate grid search for ResNet-18 trained on CIFAR-10. AGD attained performance comparable to the best tuned performance of Adam and SGD. In the **right panel**, AGD trained ResNet-50 on ImageNet to a top-1 test accuracy of 65.5%. The ImageNet baseline is SGD with a learning rate of 0.1 and no learning rate decay schedule.

experiments use square loss except ImageNet which used cross-entropy loss. Cross-entropy loss has been found to be superior to square loss for datasets with a large number of classes [156, 157].

Our experimental results are spread across five figures, four of which appear in the appendix:

- Fig. 7.3 presents some highlights of our results: First, AGD can train networks that Adam and SGD with default hyperparameters cannot. Second, for ResNet-18 on CIFAR-10, AGD attained performance comparable to the best-tuned performance of Adam and SGD. And third, AGD scales up to ImageNet, although the training accuracy improves more slowly than for SGD.

- Fig. 7.4 displays the breadth of our experiments: from training a 16-layer fully-connected network on CIFAR-10 to training ResNet-50 on ImageNet. Adam's learning rate was tuned over the logarithmic grid $\{10^{-5}, 10^{-4}, ..., 10^{-1}\}$ while for ImageNet we used a default learning rate of 0.1 for SGD without any manual

decay. AGD and Adam performed almost equally well on the depth-16, width-512 fully-connected network: 52.7% test accuracy for AGD compared to 53.5% for Adam. For ResNet-18 on CIFAR-10, Adam attained 92.9% test accuracy compared to AGD's 91.2%. On this benchmark, a fully-tuned SGD with learning rate schedule, weight decay, cross-entropy loss and bias and affine parameters can attain 93.0% test accuracy [158]. For VGG-16 on CIFAR-100, AGD achieved 67.4% test accuracy compared to Adam's 69.7%. Finally, on ImageNet, AGD achieved a top-1 test accuracy of 65.5% after 350 epochs. This is better than the test accuracy of SGD without weight decay or learning rate decay, but worse than "fully-loaded" SGD.

- Fig. 7.5 compares AGD to Adam and SGD for training an eight-layer fully-connected network of width 256. Adam and SGD's learning rates were tuned over the logarithmic grid $\{10^{-5}, 10^{-4}, ..., 10^{-1}\}$. Adam's optimal learning rate of 10^{-4} was three orders of magnitude smaller than SGD's optimal learning rate of 10^{-1}. SGD did not attain as low of an objective value as Adam or AGD.

- Fig. 7.6 shows that AGD can train FCNs with widths ranging from 64 to 2048 and depths from 2 to 32 and Fig. 7.7 shows that AGD successfully trains a four-layer FCN at varying batch size of 32 to 4096.

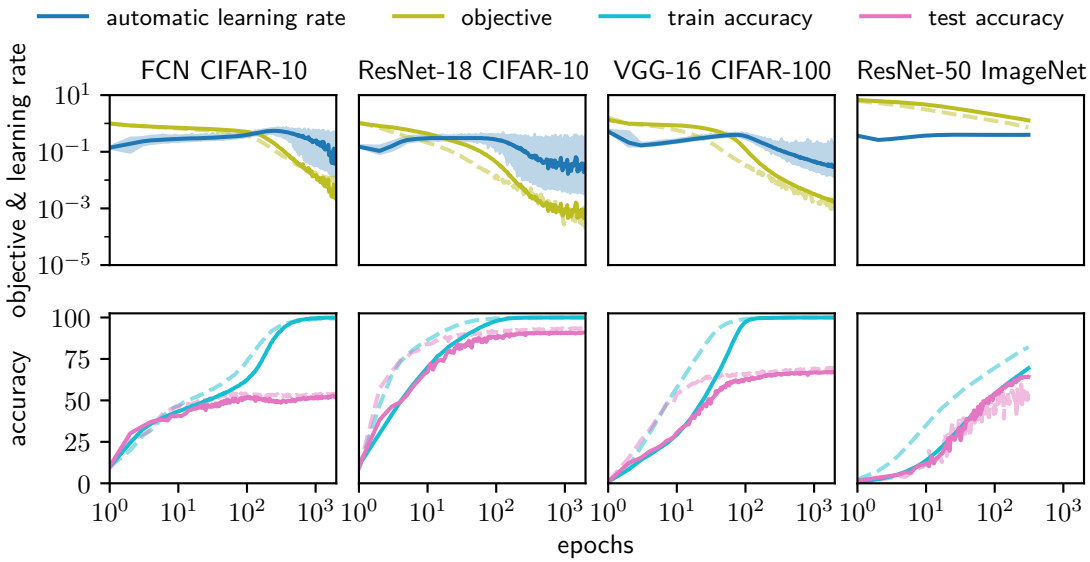


Figure 7.4: Benchmarking automatic gradient descent on a range of architectures and datasets. Solid lines are AGD and faint dashed lines are tuned Adam except for ImageNet where the dashed line is SGD with a fixed learning rate of 0.1. ImageNet used cross-entropy loss with a mini-batch size of 1024. The other experiments used square loss with a mini-batch size of 128. The **top row** plots the automatic learning rate (η in the main text) and objective value. The maximum and minimum learning rate for each epoch is included in addition to the mean for the first three plots. The **bottom row** shows the train and test accuracy.

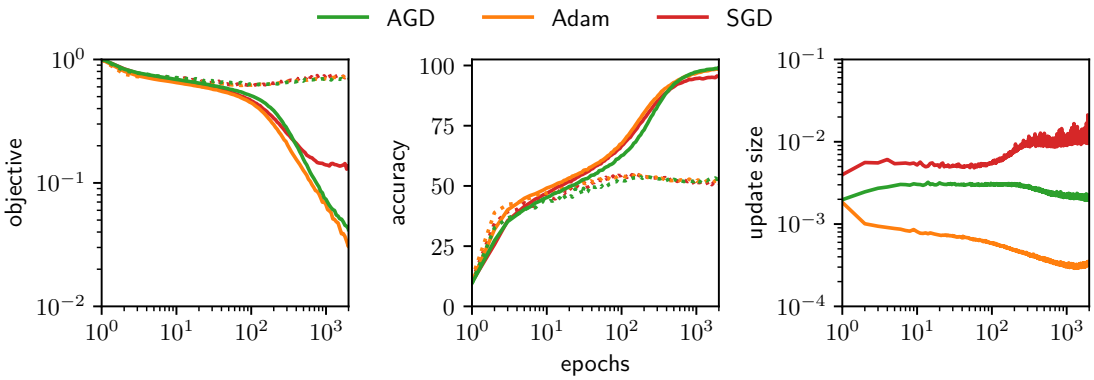


Figure 7.5: Comparing automatic gradient descent to tuned Adam and SGD. An eight-layer fully-connected network was trained on CIFAR-10 with square loss. Dotted lines show test and solid lines show train performance. The **left panel** shows the objective value: AGD and Adam attained a smaller training objective than SGD. The **middle panel** shows train and test accuracies. The **right panel** shows the relative update size averaged over layers: $\frac{1}{L}\sum_{k=1}^{L} \|\Delta \boldsymbol{W}_k\|_F / \|\boldsymbol{W}_k\|_F$. We plot the maximum, minimum and mean over an epoch.

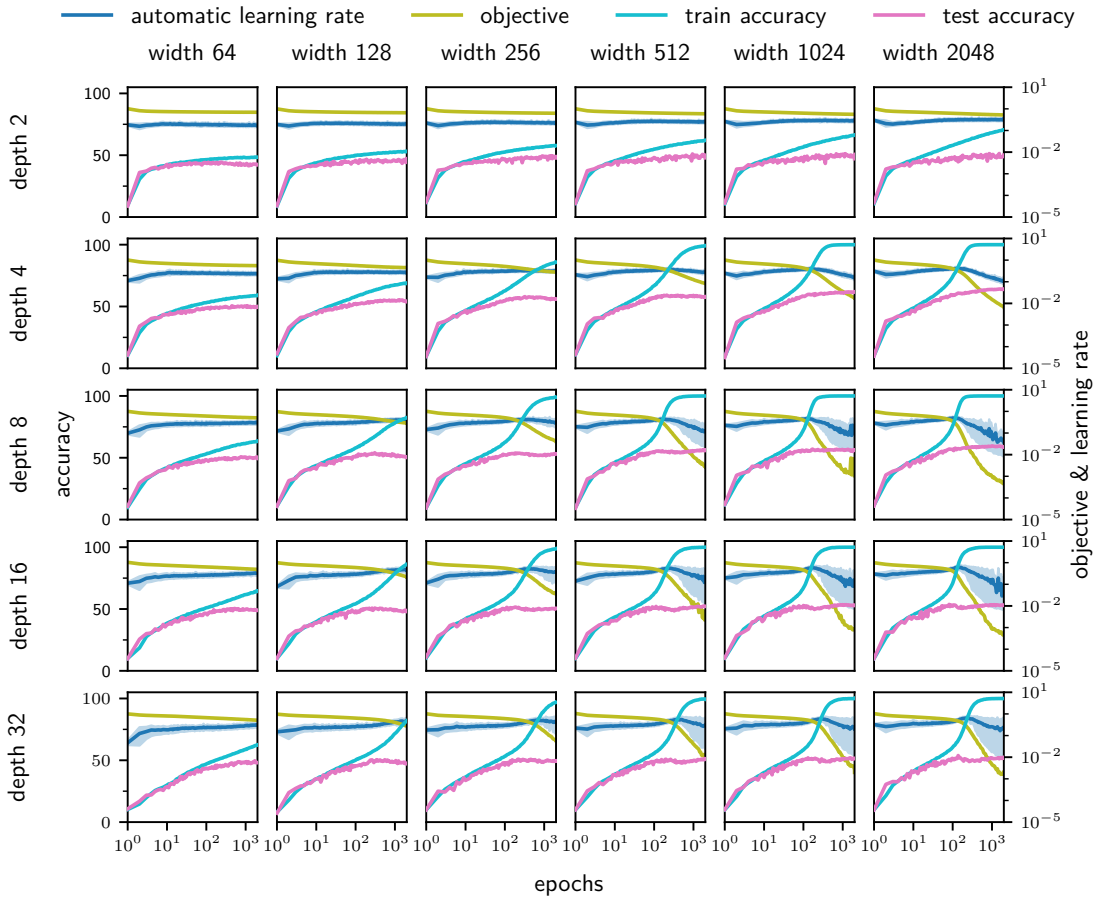


Figure 7.6: Benchmarking automatic gradient descent on networks of varying width and depth. We trained fully-connected networks on CIFAR-10 with square loss and a mini-batch size of 128. The depth ranged from 2 to 32, and the width from 64 to 2048, in powers of two. In terms of training performance, wider was always better, while depth 8 and depth 16 were superior to depth 32. In terms of test accuracy, the best performance was achieved at depth 4 and width 2048: 63.7%. The worst test performance was achieved by the smallest network of depth 2 and width 64: 42.55%. Larger networks display two broadly distinct phases of training: the automatic learning rate increases slowly while the objective decreases slowly, followed by a rapid decrease in the automatic learning rate and objective. This second phase typically coincides with reaching 100% training accuracy. See Fig. 7.5 for a comparison between Adam, SGD and AGD for the 256-width 8-layer FCN.

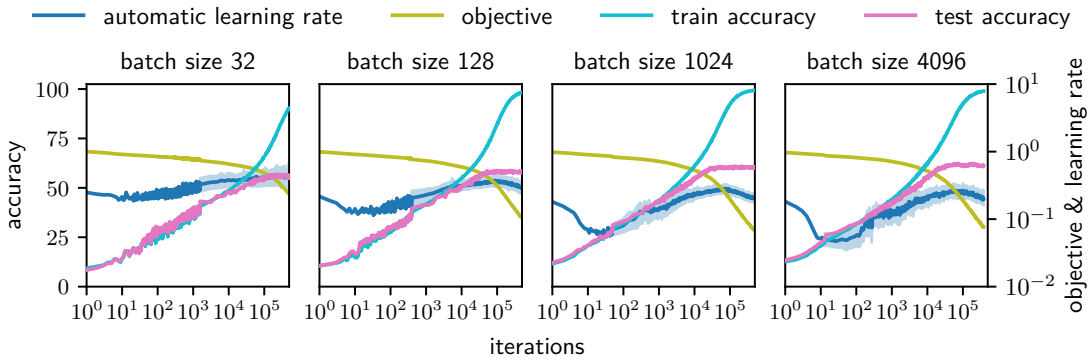


Figure 7.7: Benchmarking automatic gradient descent at varying mini-batch size. We trained four-layer fully-connected networks on CIFAR-10. The mini-batch size ranged from 32 to 4096. Test accuracy generally improved with increasing mini-batch size: the final test accuracies, in order of increasing mini-batch size, were 55.0%, 58.0%, 60.0% and 59.8%. The automatic learning rate seemed to initially dip, and this effect was more pronounced for larger mini-batch sizes. Metrics were computed every iteration during the first epoch and once per epoch from thereon—this explains the kinks visible in the plots.

7.4 Remarks

The central thesis of the work presented in this chapter is that by carefully analysing the structure of a deep learning problem—its architecture, data, and objective function—one can derive optimisation algorithms that do not require manual hyperparameter tuning [4]. The Automatic Gradient Descent (AGD) algorithm was a direct attempt to realise this vision, proposing a single, automatic learning rate derived from first principles.

Follow-up work has shifted the focus from the goal of complete hyperparameter elimination to the more general and practical goal of hyperparameter invariance. Large et al. [159] did not aim to derive a single, universally correct learning rate, but rather to construct a framework wherein a single set of hyperparameters (e.g. $O(1)$ learning rate) remains near-optimal across arbitrary network scales, from shallow, narrow models to deep, wide ones.

AGD relied on certain assumptions, such as the gradient having a stable rank of one (allowing approximating spectral norms with Frobenius norms), and derived the learning rate by assuming linear activations. The newer "Modular Norm" framework from [159] relaxes many of these assumptions. It normalises updates by directly

approximating the spectral norm for each layer, and reintroduces hyperparameters such as momentum and weight decay. The focus of the work is on carefully deriving the correct norm for each module (e.g. a ResNet block), correctly determining the gradient update as a result,

The practical results are significant, leading to marked improvements in both generalisation and convergence speed compared to the original AGD formulation. The computational overhead is manageable in practice, as the spectral norm can be efficiently estimated using methods like online power iteration.

8 Conclusion

The goal of this thesis was to contribute to our understanding of generalisation and optimisation in neural networks, specifically, the following two questions:

1. Why can highly expressive neural networks learn functions that generalise?
2. Optimiser hyperparameters can significantly affect generalisation. Why? How can we derive optimisers in a principled manner to avoid catastrophic failure?

We will briefly summarise the progress made towards answering these questions.

8.1 Part I: A toy model

In Part I, we presented a toy model of a discrete fully-connected network (DFCN) learning Boolean functions to provide an exact, end-to-end picture of generalisation, with a fully Bayesian optimisation algorithm. The core contribution was a proven one-to-one correspondence between the DFCN architecture and Disjunctive Normal Form (DNF) logical expressions. This bijection established an interpretable, tunable complexity measure, $K_{DNF}(f)$ (the length of the shortest DNF), mapping directly to the network's minimum weight norm. We showed that randomly initialised DFCNs possess a strong inductive bias, where the prior probability over functions, $P(f)$, is exponentially biased towards simple functions with low $K_{DNF}(f)$. Using a

Bayesian Monte Carlo training algorithm, we demonstrated how this prior predicts generalisation: functions with low complexity were learnable, while functions with high $K_{DNF}(f)$ were not. Finally, we showed that weight decay sharpened this simplicity bias, acting as an explicit penalty on $K_{DNF}(f)$ that promoted the learning of the minimum DNF representation – equivalent to having better features – and significantly improved generalisation.

End-to-end models are very important for understanding a topic in full, and give us confidence in our understanding of the real system. The next most complete example is probably modelling deep neural networks as kernel methods, decomposing into eigenmodes and calculating the generalisation error from this: large eigenmodes are learned faster than small ones, so provided 1) there are a small number of large eigenfunctions which 2) align well with the target, generalisation will be good [98]. However, this model does not explain why either of those conditions are met for real-world data for the neural network kernels, nor does it take into account feature learning. On the other hand, our model works on one discrete dataset, and does not permit exact error calculations, but does allow for a greater degree of interpretability.

Limitations and future directions. While the toy model is primarily intended as a source of intuition, some further exploration may be interesting. First, we know from the no free lunch theorems that if we generalise better than random for some functions, we must generalise worse than random for others. To see this linked directly to complexity with parity is striking. We can tune the learnability by tuning l-parity – a more thorough investigation into the learning curves as a function of training data m and l may be interesting. Second, the role of parameterisation and width deserves a more thorough treatment. In particular, studying how the induced function prior $P(f)$ and the effective penalty on $K_{DNF}(f)$ evolve with hidden-layer width and weight parameterisation (e.g., tied vs. untied weights, norm constraints) could illuminate transitions between kernel-like behaviour and genuine feature learning. Scaling-limit analyses (with n, input dimension, and width taken jointly to infinity) and controlled perturbations of the architecture that enable

literal reparametrisations may provide principled bridges to continuous networks, while making explicit the conditions under which our discrete insights transfer.

8.2 Part II: Generalisation

Part II addressed the first question, of generalisation, by abstracting the optimiser away (by using simple Bayesian approximations instead). We developed a quantitative account of inductive bias in deep networks, starting from the principle that understanding randomly initialised neural networks on discrete functions would provide useful insight into the properties of trained neural networks, continuing the line of work begun by Valle-Pérez et al. [64].

In Chapter 4 we analysed the prior of continuous neural networks on the Boolean data. We found that the prior $P(f) \lesssim 2^{-aK(f)}$ could be controllably weakened by tuning the initial weight variance, σ_w to move the network from the ordered regime ($a = 1$) into the chaotic regime (where $a < 1$). When $a < 1$, the prior could not counteract the 2^K growth in functions, which led to poor generalisation. This ties back to arguments from Solomonoff Induction (Section 2.3), where the universal prior probability decays as $2^{-K(f)}$. However, while this did provide useful intuition, we needed to formalise this. We can also sketch the following relation: if $P(f) \sim 2^{-aK(f)}$ for some suitable complexity measure K and as the number of functions (and thus the rank of the function $R(f)$) grows as 2^K, then $P(f) \sim R(f)^{-a}$. When $a = 1$, the relationship between the probability and rank would satisfy Zipf's law.

In Chapter 5 we show that, indeed, a wide variety of modern architectures (FCNs, CNNs, ResNets, Transformers, etc.) all exhibit a Zipf's law in their function prior, over many orders of magnitude. We were able to prove that the priors of all NNGPs will be Zipfian for a sufficiently great eigenvalue decay in the kernel. We were then able to show that networks which generalise well operate at this critical Zipfian point. Deviating from it harms learning: if the prior is too flat ($a < 1$), the model has too little bias and we get errors from variance that do not decay to 0 even as the amount of data $n \to \infty$. Conversely, if the prior is too steep ($a > 1$), the model will underfit, placing overly high probability on a narrow set of simple

functions (unless the likelihood is perfectly hard). This result formalises a balance that the deep learning community only informally understood before.

The discrete-functions viewpoint adopted here is relatively uncommon, yet it yields a compact and predictive account of inductive bias that complements existing kernel–eigenmode analyses. In particular, it offers an alternative perspective to Canatar et al. [98], Cui et al. [100]: whereas the spectral approach quantifies learnability via alignment and eigenvalue decay of the (linearised) kernel, our Zipfian prior organises hypotheses by description length and predicts generalisation from the resulting probability–rank law.

Limitations and future directions. The main limitations of this section are twofold. First, the explicit finite-sample error characterisations remain incomplete: although the probability–rank relation explains the qualitative transition between under- and over-regularised regimes, sharper nonasymptotic bounds (e.g., in terms of n, input dimension, and the effective Zipf exponent) would strengthen the story. Second, the treatment of feature learning is only indirect. We verify Zipfian behaviour empirically in finite networks that do learn features, but we do not yet provide a theoretical account of how representation dynamics preserve—or distort—the Zipf law away from the NNGP/NTK limits.

We see two promising directions. (i) A finite-width theory that tracks how the induced function prior deforms during training (e.g., via early-to-late time linearisation, path-norm or margin-based surrogates for complexity) could bridge the gap between fixed-kernel Zipfian priors and fully adaptive representations. (ii) A synthesis with the spectral view that links prior Zipf exponents to kernel eigenvalue tails and target–eigenfunction alignment may yield testable equivalences: e.g., conditions under which Zipfian criticality implies fast learning of a small number of aligned modes, and conversely. Establishing such correspondences would turn the present qualitative match into formal guarantees and clarify when simplicity-driven and alignment-driven accounts make divergent predictions.

8.3 Part III: Optimisation

Part III addressed the second question, of optimisation. In the toy model from Part I, we trained a discrete neural network with an MCMC sampler and weight decay. For this model, we could explain, end-to-end, how an explicit ℓ_2 penalty sharpened the simplicity bias of the induced function prior. In Part II we then abstracted away the optimiser altogether to study the inductive bias of randomly initialised networks via their function priors. The final part sought to understand what happens when you add the optimiser back in, by analysing how concrete optimisation dynamics interact with representation learning, and by deriving a principled, architecture-aware algorithm.

In Chapter 6 we first established that, for sufficiently wide networks and suitably simple tasks, the training dynamics of stochastic gradient descent (SGD) approximate Bayesian inference. We quantified the regime in which this approximation holds and identified the mechanisms by which it breaks: as width decreases, depth increases, or data/task complexity grows, networks depart from lazy/linearised behaviour and begin to learn features not present at initialisation. Empirically, we observed such feature learning even in modest settings (e.g., CNNs on CIFAR-10), with measurable divergences between posterior predictions implied by the initial kernel and the trajectories followed by SGD. To make these observations systematic, we introduced a framework for quantifying feature learning that disentangles changes in coarse-grained measures like generalisation error due to (i) reweighting fixed features versus (ii) genuine representation change. Applying this framework across optimisers and architectures revealed consistent patterns: optimiser hyperparameters do affect the degree of representation learning.

Finally, in Chapter 7 we moved from analysis to design. We proposed *Automatic Gradient Descent* (AGD), a first-order, hyperparameter-free optimiser derived from an extension of mirror descent that explicitly incorporates neural architecture. The derivation proceeds by constructing a tractable majorisation of the loss in function space and selecting, at each iteration, the update that minimises this

majorisation. The resulting parameterisation and rule normalise each layer's update by appropriate scale factors (e.g., layer norms and spectral radii) in a manner similar to μP [41]. Practically, AGD trains fully connected and convolutional networks at ImageNet scale with a single default configuration, while theoretical analysis provides convergence guarantees under stated conditions. Empirically, AGD attains competitive accuracy—typically within a few percentage points of carefully tuned SGD/Adam—without any learning-rate tuning, and it remains stable for very deep and very wide models, promoting feature learning rather than collapsing into the lazy regime.

Limitations and future work. Our analysis of the SGD–Bayes correspondence remains asymptotic and without precise empirical results. Sharper finite-sample characterisations of when and how the approximation fails would be valuable, especially as a function of width, depth, and data complexity. The feature-learning quantification isolates representation change; however, a full theory linking these metrics to generalisation across realistic distributions remains open. Furthermore, extending the theory to the actual intermediate layers would be a significant improvement. Each layer aims to align to the requirements of the next, rather than the target function, so a proper measure of intermediate layer feature learning would take this into account, rather than just using the target.

For Chapter 7, while Automatic Gradient Descent (AGD) represents an innovative step toward eliminating manual hyperparameter tuning in deep learning optimisation, it carries several theoretical and practical limitations. Theoretically, AGD's guarantees depend on a set of structural assumptions concerning smoothness, conditioning, and the maintenance of certain norm bounds throughout training. These assumptions are useful for constructing a tractable smoothness model, but they are not proven to hold in general neural network training dynamics. Consequently, while the derivations are mathematically consistent under these assumptions, their real-world relevance is uncertain, limiting the generality of AGD's theoretical convergence claims.

However, this paper did lay the foundations for further work (especially using the spectral norm-inspired parameterisation in Yang et al. [41]). Later work, such as Large et al. [159], has built on AGD's parameterisation and update rule by introducing "modular norm", a more expressive alternative that removes the Frobenius-norm approximation used in AGD and aligns more closely with the spectral norm geometry of the network. This refinement delivers significant empirical gains and shows how AGD's framework can evolve toward more faithful representations of neural architecture. Finally, no automatic optimiser or tuning-free method is likely to outperform an optimally tuned learning rate: a well-calibrated learning rate can often squeeze out the final few per cent of performance that matters in competitive or large-scale applications. AGD therefore represents a valuable conceptual framework for architecture-aware optimisation, but one whose simplifying assumptions and lack of fine-grained control currently limit its practical utility.

Bibliography

[1] Chris Mingard, Joar Skalse, Guillermo Valle-Pérez, David Martínez-Rubio, Vladimir Mikulik, and Ard A Louis. Neural networks are a priori biased towards boolean functions with low entropy. *arXiv preprint arXiv:1909.11522*, 2019.

[2] Chris Mingard, Guillermo Valle-Pérez, Joar Skalse, and Ard A Louis. Is sgd a bayesian sampler? well, almost. *Journal of Machine Learning Research*, 22(79): 1–64, 2021.

[3] Yizhang Lou, Chris E Mingard, and Soufiane Hayou. Feature learning and signal propagation in deep neural networks. In *International Conference on Machine Learning*, pages 14248–14282. PMLR, 2022.

[4] Jeremy Bernstein, Chris Mingard, Kevin Huang, Navid Azizan, and Yisong Yue. Automatic gradient descent: Deep learning without hyperparameters. *arXiv preprint arXiv:2304.05187*, 2023.

[5] Chris Mingard, Henry Rees, Guillermo Valle-Pérez, and Ard A Louis. Do deep neural networks have an inbuilt occam's razor? *arXiv preprint arXiv:2304.06670*, 2023.

[6] Yoonsoo Nam, Nayara Fonseca, Seok Hyeong Lee, Chris Mingard, Ard Louis, et al. An exactly solvable model for emergence and scaling laws in the multitask sparse parity problem. *Advances in Neural Information Processing Systems*, 37: 39632–39693, 2024.

[7] Chris Mingard, Jessica Pointing, Charles London, Yoonsoo Nam, and Ard A Louis. Exploiting the equivalence between quantum neural networks and perceptrons. *arXiv preprint arXiv:2407.04371*, 2024.

[8] Yoonsoo Nam, Chris Mingard, Seok Hyeong Lee, Soufiane Hayou, and Ard Louis. Visualising feature learning in deep neural networks by diagonalizing the forward feature map. *arXiv preprint arXiv:2410.04264v1*, 2024. URL https://arxiv.org/abs/2410.04264v1.

[9] Chris Mingard, Lukas Seier, Niclas Göring, Andrei-Vlad Badelita, Charles London, and Ard Louis. Characterising the inductive biases of neural networks on boolean data. *arXiv preprint arXiv:2505.24060*, 2025.

[10] Niclas Alexander Göring, Charles London, Abdurrahman Hadi Erturk, Chris Mingard, Yoonsoo Nam, and Ard A Louis. Feature learning is decoupled from generalization in high capacity neural networks. In *High-dimensional Learning Dynamics 2025*, 2025.

[11] Chris Mingard, Sean Ridout, Radoslaw Grabarczyk, Charles London, Kamal Dingle, Ilya Nemenman, and Ard A. Louis. Successful high capacity machine learning models exhibit a universal zipf's law. To be submitted to journals some time in late 2025, 2025.

[12] Niclas Göring, Chris Mingard, Yoonsoo Nam, and Ard A. Louis. A simple mean field model of feature learning. Submitted to ICLR 2026, 2025. URL https://openreview.net/pdf?id=FVQzqSIJcC.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[15] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359, 2017.

[16] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

[17] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon Kohl, Andrew Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583–589, 07 2021. doi: 10.1038/s41586-021-03819-2.

[18] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

[19] Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. Gemini 2.0 flash. https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/#gemini-2-0-flash, dec 2024.

[20] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*, 2024.

[21] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[22] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[23] Ilya Sutskever, Rafal Jozefowicz, Karol Gregor, Danilo Rezende, Tim Lillicrap, and Oriol Vinyals. Towards principled unsupervised learning. *arXiv preprint arXiv:1511.06440*, 2015.

[24] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006.

[25] Gregory J Chaitin. On the simplicity and speed of programs for computing infinite sets of natural numbers. *Journal of the ACM (JACM)*, 16(3):407–422, 1969.

[26] L.A. Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, 10(3):30–35, 1974.

[27] Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.

[28] M. Li and P.M.B. Vitanyi. *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag New York Inc, 2008.

[29] Shayle R Searle. *Linear models*. John Wiley & Sons, 1997.

[30] Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation. *arXiv preprint arXiv:2105.14368*, 2021.

[31] Ernest K Ryu and Stephen Boyd. Primer on monotone operator methods. *Appl. comput. math*, 15(1):3–43, 2016.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436, 2015.

[33] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[35] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[36] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[37] Diederik P Kingma and Jimmy Lei Ba. Adam: Amethod for stochastic optimization. In *Proc. 3rd Int. Conf. Learn. Representations*, 2014.

[38] Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum. *Advances in Neural Information Processing Systems*, 33: 18261–18271, 2020.

[39] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

[40] Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pages 11727–11737. PMLR, 2021.

[41] Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

[42] Greg Yang and Edward J Hu. Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2011.14522*, 2020.

[43] Yoonsoo Nam, Seok Hyeong Lee, Clementine CJ Domine, Yeachan Park, Charles London, Wonyl Choi, Niclas Goring, and Seungjai Lee. Position: Solve layerwise linear models first to understand neural dynamical phenomena (neural collapse, emergence, lazy/rich regime, and grokking). *arXiv preprint arXiv:2502.21009*, 2025.

[44] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in neural information processing systems*, 32, 2019.

[45] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[46] Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.

[47] Katie Everett, Lechao Xiao, Mitchell Wortsman, Alexander A. Alemi, Roman Novak, Peter J. Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12666–12700. PMLR, 2024. doi: 10.48550/arXiv.2407.05872.

[48] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[49] Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. In *Advances in Neural Information Processing Systems*, pages 9951–9960, 2019.

[50] Gadi Naveh, Oded Ben David, Haim Sompolinsky, and Zohar Ringel. Predicting the outputs of finite deep neural networks trained with noisy gradients. *Physical Review E*, 104(6):064301, 2021.

[51] Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018.

[52] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *arxiv*, 2020(12):124002, 2020. ISSN 1742-5468. doi: 10.1088/1742-5468/abc62b. URL http://arxiv.org/abs/1902.06720.

[53] Guillermo Ortiz-Jiménez, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. What can linearized neural networks actually say about generalization?, 2021. URL https://arxiv.org/abs/2106.06770.

[54] Benjamin Bowman and Guido Montufar. Spectral bias outside the training set for deep networks in the kernel regime. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

[55] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33), July 2018. ISSN 1091-6490. doi: 10.1073/pnas.1806579115. URL http://dx.doi.org/10.1073/pnas.1806579115.

[56] Clémentine C. J. Dominé, Nicolas Anguita, Alexandra M. Proca, Lukas Braun, Daniel Kunin, Pedro A. M. Mediano, and Andrew M. Saxe. From lazy to rich: Exact learning dynamics in deep linear networks, 2025. URL https://arxiv.org/abs/2409.14623.

[57] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1 (1):12, 2021.

[58] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

[59] Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

[60] Zvi Kohavi and Niraj K. Jha. *Switching and finite automata theory, third edition*, volume 9780521857482. Cambridge University Press, United Kingdom, January 2009. ISBN 9780521857482. doi: 10.1017/CBO9780511816239. Publisher Copyright: © The McGraw-Hill Companies Inc.and © Z. Kohavi and N. Jha 2010.

[61] W. V. Quine. The problem of simplifying truth functions. *American Mathematical Monthly*, 59:521–531, 1952. URL https://api.semanticscholar.org/CorpusID:124965557.

[62] Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael Saks. Minimizing DNF formulas and AC0 circuits given a truth table. *Electronic Colloquium on Computational Complexity (ECCC)*, 01 2005.

[63] Guillermo Valle-Pérez and Ard A Louis. Generalization bounds for deep learning. *arXiv preprint arXiv:2012.04115*, 2020.

[64] Guillermo Valle-Pérez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *arXiv preprint arXiv:1805.08522*, 2018.

[65] Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature Communications*, 9(1):1–7, 2018.

[66] Kamaludin Dingle, Guillermo Valle Pérez, and Ard A Louis. Generic predictions of output probability based on complexities of inputs and outputs. *Scientific Reports*, 10(1):1–9, 2020.

[67] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/hash/d8330f857a17c53d217014ee776bfd50-Abstract.html.

[68] Fengxiang He, Tongliang Liu, and Dacheng Tao. Why resnet works? residuals generalize. *IEEE Transactions on Neural Networks and Learning Systems*, PP: 1–14, 02 2020. doi: 10.1109/TNNLS.2020.2966319.

[69] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets, 2018. URL https://arxiv.org/abs/1712.09913.

[70] Hugo Tessier, Vincent Gripon, Mathieu Léonardon, Matthieu Arzel, Thomas Hannagan, and David Bertrand. Rethinking weight decay for efficient neural network pruning. *Journal of Imaging*, 8(3):64, March 2022. ISSN 2313-433X. doi: 10.3390/jimaging8030064. URL http://dx.doi.org/10.3390/jimaging8030064.

[71] Satwik Bhattamishra, Arkil Patel, Varun Kanade, and Phil Blunsom. Simplicity bias in transformers and their ability to learn sparse boolean functions. *arXiv preprint arXiv:2211.12316*, 2022.

[72] David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S. Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, Aaron Courville, Simon Lacoste-Julien, and Yoshua Bengio. A closer look at memorization in deep networks. Proceedings of the 34th International Conference on Machine Learning (ICML'17), 2017. URL https://arxiv.org/abs/1706.05394.

[73] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

[74] Giacomo De Palma, Bobak Toussi Kiani, and Seth Lloyd. Random deep neural networks are biased towards simple functions. *arXiv preprint arXiv:1812.10156*, 2018.

[75] Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. SGD on neural networks learns functions of increasing complexity. *Advances in neural information processing systems*, 32, 2019.

[76] Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.

[77] Alexander R Farhang, Jeremy D Bernstein, Kushal Tirumala, Yang Liu, and Yisong Yue. Investigating generalization by controlling normalized margin. In *International Conference on Machine Learning*, pages 6324–6336. PMLR, 2022.

[78] Ping-yeh Chiang, Renkun Ni, David Yu Miller, Arpit Bansal, Jonas Geiping, Micah Goldblum, and Tom Goldstein. Loss landscapes are all you need: Neural network generalization can be explained without the implicit bias of gradient descent. In *The Eleventh International Conference on Learning Representations*, 2023.

[79] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368, 2016.

[80] Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. *arXiv preprint arXiv:1611.01232*, 2017.

[81] Greg Yang and Hadi Salman. A fine-grained spectral perspective on neural networks. *arXiv preprint arXiv:1907.10599*, 2019.

[82] Cullen Schaffer. A conservation law for generalization performance. In *Machine Learning Proceedings 1994*, pages 259–265. Elsevier, 1994.

[83] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[84] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[85] David A McAllester. Some PAC-Bayesian theorems. *Machine Learning*, 37(3): 355–363, 1999.

[86] Leo Breiman. Reflections after refereeing papers for nips. In *The Mathematics of Generalization*, pages 11–15. Addison-Wesley, 1995.

[87] Lenka Zdeborová. Understanding deep learning is also a job for physicists. *Nature Physics*, pages 1–3, 2020.

[88] Stefano Spigler, Mario Geiger, and Matthieu Wyart. Asymptotic learning curves of kernel methods: empirical data versus teacher–student paradigm. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(12):124001, 2020.

[89] Sebastian Goldt, Marc Mézard, Florent Krzakala, and Lenka Zdeborová. Modelling the influence of data structure on learning in neural networks. *arXiv preprint arXiv:1909.11500*, 2019.

[90] Tom M Mitchell. The need for biases in learning generalizations (rutgers computer science tech. rept. cbm-tr-117). *Rutgers University*, 1980.

[91] Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.

[92] Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6):1076–1136, 2011.

[93] Jordi Grau-Moya, Tim Genewein, Marcus Hutter, Laurent Orseau, Grégoire Delétang, Elliot Catt, Anian Ruoss, Li Kevin Wenliang, Christopher Mattern, Matthew Aitchison, et al. Learning universal predictors. *arXiv preprint arXiv:2401.14953*, 2024.

[94] Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, et al. Language modeling is compression. *arXiv preprint arXiv:2309.10668*, 2023.

[95] Ziguang Li, Chao Huang, Xuliang Wang, Haibo Hu, Cole Wyeth, Dongbo Bu, Quan Yu, Wen Gao, Xingwu Liu, and Ming Li. Understanding is compression. *arXiv preprint arXiv:2407.07723*, 2024.

[96] Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum dependent learning curves in kernel regression and wide neural networks. In *International Conference on Machine Learning*, pages 1024–1034. PMLR, 2020.

[97] Omry Cohen, Or Malka, and Zohar Ringel. Learning curves for overparametrized deep neural networks: A field theory perspective. *Physical Review Research*, 3(2): 023034, 2021.

[98] Abdulkadir Canatar, Blake Bordelon, and Cengiz Pehlevan. Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks. *Nature communications*, 12(1):1–12, 2021.

[99] Ouns El Harzli, Guillermo Valle-Pérez, and Ard A Louis. Double-descent curves in neural networks: a new perspective using gaussian processes. *arXiv preprint arXiv:2102.07238*, 2021.

[100] Hugo Cui, Bruno Loureiro, Florent Krzakala, and Lenka Zdeborová. Generalization error rates in kernel regression: The crossover from the noiseless to noisy regime. *Advances in Neural Information Processing Systems*, 34:10131–10143, 2021.

[101] James B Simon, Madeline Dickens, and Michael R DeWeese. Neural tangent kernel eigenvalues accurately predict generalization. *arXiv preprint arXiv:2110.03922*, 2021.

[102] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.

[103] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 810:1–124, 2019.

[104] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019.

[105] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[106] Cullen Schaffer. Overfitting avoidance as bias. *Machine learning*, 10(2):153–178, 1993.

[107] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[108] David J Schwab, Ilya Nemenman, and Pankaj Mehta. Zipf's law and criticality in multivariate data without fine-tuning. *Physical review letters*, 113(6):068102, 2014.

[109] Laurence Aitchison, Nicola Corradi, and Peter E Latham. Zipf's law arises naturally when there are underlying, unobserved variables. *PLoS computational biology*, 12(12):e1005110, 2016.

[110] Vudtiwat Ngampruetikorn, Ilya Nemenman, and David J Schwab. Extrinsic vs intrinsic criticality in systems with many components. *Physical Review Research*, 7 (1):013188, 2025.

[111] T. Hastie, A. Montanari, S. Rosset, and R.J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *arXiv preprint: arXiv:1903.08560*, 2019.

[112] James B Simon, Madeline Dickens, Dhruva Karkada, and Michael R DeWeese. The eigenlearning framework: A conservation law perspective on kernel regression and wide neural networks. *arXiv preprint arXiv:2110.03922*, 2021.

[113] Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070, 2020.

[114] William Bialek, Ilya Nemenman, and Naftali Tishby. Predictability, complexity, and learning. *Neural computation*, 13(11):2409–2463, 2001.

[115] Jeremy Bernstein, Alex Farhang, and Yisong Yue. Kernel interpolation as a Bayes point machine. *arXiv*, 2021.

[116] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[117] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[118] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[119] Adityanarayanan Radhakrishnan. Lecture 4: NNGP, dual activations, and over-parameterization, Jan 2022. URL https://web.mit.edu/modernml/course/lectures/MLClassLecture4.pdf.

[120] Alex J Smola and Bernhard Schölkopf. On a kernel-based method for pattern recognition, regression, approximation, and operator inversion. *Algorithmica*, 22: 211–231, 1998.

[121] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[122] Nello Cristianini, John Shawe-Taylor, Andre Elisseeff, and Jaz Kandola. On kernel-target alignment. *Advances in neural information processing systems*, 14, 2001.

[123] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. *Advances in neural information processing systems*, 30, 2017.

[124] Abdulkadir Canatar and Cengiz Pehlevan. A kernel analysis of feature learning in deep neural networks. In *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1–8. IEEE, 2022.

[125] Arthur Jacot, Berfin Simsek, Francesco Spadaro, Clément Hongler, and Franck Gabriel. Kernel alignment risk estimator: Risk prediction from training data. *Advances in Neural Information Processing Systems*, 33:15568–15578, 2020.

[126] James B Simon, Madeline Dickens, and Michael R DeWeese. A theory of the inductive bias and generalization of kernel regression and wide neural networks. *arXiv e-prints*, pages arXiv–2110, 2021.

[127] Mark O Hill. Diversity and evenness: a unifying notation and its consequences. *Ecology*, 54(2):427–432, 1973.

[128] Vardan Papyan, XY Han, and David L Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

[129] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMLR, 2019.

[130] Aristide Baratin, Thomas George, César Laurent, R Devon Hjelm, Guillaume Lajoie, Pascal Vincent, and Simon Lacoste-Julien. Implicit regularization via neural feature alignment. In *International Conference on Artificial Intelligence and Statistics*, pages 2269–2277. PMLR, 2021.

[131] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[132] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[133] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[134] Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(26):e2311878121, 2024. doi: 10.1073/pnas.2311878121.

[135] Mario Geiger, Leonardo Petrini, and Matthieu Wyart. Landscape and training regimes in deep learning. *Physics Reports*, 924:1–18, 2021.

[136] Nikhil Vyas, Alexander Atanasov, Blake Bordelon, Depen Morwani, Sabarish Sainathan, and Cengiz Pehlevan. Feature-learning networks are consistent across widths at realistic scales. *arXiv preprint arXiv:2305.18411*, 2023.

[137] Tomer Galanti, Zachary S Siegel, Aparna Gupte, and Tomaso Poggio. SGD and weight decay provably induce a low-rank bias in neural networks. *arXiv preprint arXiv:2206.05794*, 2022.

[138] Grégoire Montavon, Mikio L Braun, and Klaus-Robert Müller. Kernel analysis of deep networks. *Journal of Machine Learning Research*, 12(78):2563–2581, 2011.

[139] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[140] Inbar Seroussi, Gadi Naveh, and Zohar Ringel. Separation of scales and a thermodynamic description of feature learning in some CNNs. *Nature Communications*, 14(1):908, 2023.

[141] Kenneth Lange. *MM Optimization Algorithms*. Society for Industrial and Applied Mathematics, 2016.

[142] Jeremy Bernstein, Arash Vahdat, Yisong Yue, and Ming-Yu Liu. On the distance between two neural networks and the stability of learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[143] Lev M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 1967.

[144] Jeremy Bernstein. *Optimisation & Generalisation in Networks of Neurons*. Ph.D. thesis, California Institute of Technology, 2022.

[145] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations*, 2014.

[146] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in neural information processing systems*, pages 8572–8583, 2019.

[147] Hermann Weyl. Das asymptotische Verteilungsgesetz der Eigenwerte linearer partieller Differentialgleichungen (mit einer Anwendung auf die Theorie der Hohlraumstrahlung). *Mathematische Annalen*, 1912.

[148] George Philipp, Dawn Xiaodong Song, and Jaime G. Carbonell. The exploding gradient problem demystified. *arXiv:1712.05577*, 2017.

[149] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32K for ImageNet training. Technical report, University of California, Berkeley, 2017.

[150] Simon Carbonnelle and Christophe De Vleeschouwer. Layer rotation: A surprisingly simple indicator of generalization in deep networks? In *ICML Workshop on Identifying and Understanding Deep Learning Phenomena*, 2019.

[151] Yonatan Loewenstein, Annerose Kuras, and Simon Rumpel. Multiplicative dynamics underlie the emergence of the log-normal distribution of spine sizes in the neocortex in vivo. *Journal of Neuroscience*, 2011.

[152] Andy Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, 2021.

[153] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. *Applied and Computational Harmonic Analysis*, 2022.

[154] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[155] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, 2009.

[156] Ahmet Demirkaya, Jiasi Chen, and Samet Oymak. Exploring the role of loss functions in multiclass classification. *Conference on Information Sciences and Systems*, 2020.

[157] Like Hui and Mikhail Belkin. Evaluation of neural architectures trained with square loss vs. cross-entropy in classification tasks. In *International Conference on Learning Representations*, 2021.

[158] Kuang Liu. Train CIFAR-10 with PyTorch. https://github.com/kuangliu/pytorch-cifar, 2017.

[159] Tim Large, Yang Liu, Minyoung Huh, Hyojin Bahng, Phillip Isola, and Jeremy Bernstein. Scalable optimization in the modular norm. *Advances in Neural Information Processing Systems*, 37:73501–73548, 2024.

[160] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

[161] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

[162] Pedro Domingos. The role of occam's razor in knowledge discovery. *Data mining and knowledge discovery*, 3(4):409–425, 1999.

[163] Karl Raimund Popper. Logik der forschung. 1989.

[164] Elliott Sober. *Ockham's razors*. Cambridge University Press, 2015.

[165] Ray J Solomonoff. A preliminary report on a general theory of inductive inference. Citeseer, 1960.

[166] Andrei Nikolaevic Kolmogorov. Three approaches to the quantitative definition of information. *International journal of computer mathematics*, 2(1-4):157–168, 1968.

[167] Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature Communications*, 9(1):761, 2018.

[168] Bruno Bauwens. Relating and contrasting plain and prefix kolmogorov complexity. *Theory of Computing Systems*, 58(3):482–501, 2016.

[169] Marcus Hutter. A theory of universal artificial intelligence based on algorithmic complexity. *arXiv preprint cs/0004001*, 2000. URL https://arxiv.org/abs/cs/0004001.

[170] Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

[171] Chris S Wallace and David M Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968.

[172] Jürgen Schmidhuber. The speed prior: a new simplicity measure yielding near-optimal computable predictions. In *International conference on computational learning theory*, pages 216–228. Springer, 2002.

[173] David H Wolpert and R Waters. The relationship between PAC, the statistical physics framework, the bayesian framework, and the vc framework. In *In*. Citeseer, 1994.

[174] Tor Lattimore and Marcus Hutter. No free lunch versus occam's razor in supervised learning. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, pages 223–235. Springer, 2013.

[175] Tom F Sterkenburg. Solomonoff prediction and occam's razor. *Philosophy of Science*, 83(4):459–479, 2016.

[176] Sven Neth. A dilemma for Solomonoff prediction. *Philosophy of Science*, pages 1–25, 2022.

[177] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[178] Stefano Spigler, Mario Geiger, and Matthieu Wyart. Asymptotic learning curves of kernel methods: empirical data versus teacher–student paradigm. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(12):124001, 2020.

[179] Matthias Hein and Jean-Yves Audibert. Intrinsic dimensionality estimation of submanifolds in rd. In *Proceedings of the 22nd international conference on Machine learning*, pages 289–296, 2005.

[180] Christopher De Sa. Lecture 12: Bias-variance tradeoff. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html, 2018.

[181] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th international conference on machine learning*, pages 231–238. Morgan Kaufmann Stanford, 2000.

[182] Pierre Geurts. Bias vs variance decomposition for regression and classification. In *Data mining and knowledge discovery handbook*, pages 733–746. Springer, 2009.

[183] Jason W Rocks and Pankaj Mehta. Memorizing without overfitting: Bias, variance, and interpolation in overparameterized models. *Physical Review Research*, 4(1): 013201, 2022.

[184] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[185] David A McAllester. PAC-Bayesian model averaging. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 164–170, 1999.

[186] Radford M Neal. Priors for infinite networks (tech. rep. no. crg-tr-94-1). *University of Toronto*, 1994.

[187] Blake Bordelon and Cengiz Pehlevan. Self-consistent dynamical field theory of kernel evolution in wide neural networks. *Advances in Neural Information Processing Systems*, 35:32240–32256, 2022.

[188] Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the chomsky hierarchy, 2023. URL https://arxiv.org/abs/2207.02098.

[189] Kamaludin Dingle, Steffen Schaper, and Ard A Louis. The structure of the genotype–phenotype map strongly constrains the evolution of non-coding rna. *Interface focus*, 5(6):20150053, 2015.

[190] Marcus Hutter, David Quarel, and Elliot Catt. *An introduction to universal artificial intelligence*. CRC Press, 2024.

[191] Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural networks and the Chomsky Hierarchy. *arXiv preprint arXiv:2207.02098*, 2022.

[192] Damien Teney, Armand Mihai Nicolicioiu, Valentin Hartmann, and Ehsan Abbasnejad. Neural redshift: Random networks are not random functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4786–4796, 2024.

[193] Damien Teney, Liangze Jiang, Florin Gogianu, and Ehsan Abbasnejad. Do we always need the simplicity bias? looking for optimal inductive biases in the wild, 2025. URL http://arxiv.org/abs/2503.10065.

[194] Greg Yang and Hadi Salman. A fine-grained spectral perspective on neural networks. *arXiv preprint arXiv:1907.10599*, 2020. doi: 10.48550/arXiv.1907.10599. URL http://arxiv.org/abs/1907.10599.

[195] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[196] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the bias-variance trade-off. *arXiv preprint arXiv:1812.11118*, 2018.

[197] Mikhail Belkin, Daniel Hsu, and Ji Xu. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science*, 2(4):1167–1180, 2020.

[198] Ronen Basri, Meirav Galun, Amnon Geifman, David Jacobs, Yoni Kasten, and Shira Kritchman. Frequency bias in neural networks for input of non-uniform density. In *Proceedings of the 37th International Conference on Machine Learning*, pages 685–694. PMLR, 2020. URL https://proceedings.mlr.press/v119/basri20a.html. ISSN: 2640-3498.

[199] Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Basri Ronen. On the similarity between the laplace and neural tangent kernels. In *Advances in Neural Information Processing Systems*, volume 33, pages 1451–1461. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1006ff12c465532f8c574aeaa4461b16-Abstract.html.

[200] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

[201] Emmanuel Abbe, Elisabetta Cornacchia, Jan Hązła, and Donald Kougang-Yombi. Learning high-degree parities: The crucial role of the initialization. In *International Conference on Learning Representations*, 2025. doi: 10.48550/arXiv.2412.04910. URL http://arxiv.org/abs/2412.04910.

[202] Amnon Geifman, Meirav Galun, David Jacobs, and Ronen Basri. On the spectral bias of convolutional neural tangent and gaussian process kernels. In *Advances in Neural Information Processing Systems*, 2022. doi: 10.48550/arXiv.2203.09255. URL http://arxiv.org/abs/2203.09255.

[203] Omry Cohen, Or Malka, and Zohar Ringel. Learning curves for deep neural networks: A gaussian field theory perspective. *arxiv*, 3(2):023034, 2021. ISSN 2643-1564. doi: 10.1103/PhysRevResearch.3.023034. URL http://arxiv.org/abs/1906.05301.

[204] Emmanuel Abbe, Enric Boix-Adsera, and Theodor Misiakiewicz. The merged-staircase property: a necessary and nearly sufficient condition for SGD learning of sparse functions on two-layer neural networks, 2024. URL http://arxiv.org/abs/2202.08658.

[205] Maria Refinetti, Sebastian Goldt, Florent Krzakala, and Lenka Zdeborová. Classifying high-dimensional gaussian mixtures: Where kernel methods fail and neural networks succeed, 2021. URL http://arxiv.org/abs/2102.11742.

[206] Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When do neural networks outperform kernel methods? In *Advances in Neural Information Processing Systems*, volume 33, pages 14820–14830. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/a9df2255ad642b923d95503b9a7958d8-Abstract.html.

[207] Konstantin Donhauser, Mingqi Wu, and Fanny Yang. How rotational invariance of common kernels prevents generalization in high dimensions. In *Proceedings of the 38th International Conference on Machine Learning*, pages 2804–2814. PMLR, 2021. URL https://proceedings.mlr.press/v139/donhauser21a.html. ISSN: 2640-3498.

[208] Blake Bordelon, Alexander Atanasov, and Cengiz Pehlevan. How feature learning can improve neural scaling laws, 2024. URL http://arxiv.org/abs/2409.17858.

[209] Daniel A Roberts, Sho Yaida, and Boris Hanin. *The principles of deep learning theory*. Cambridge University Press, 2022.

[210] Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392*, 2019.

[211] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[212] Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. *CoRR*, abs/1710.06451, 2017. URL http://arxiv.org/abs/1710.06451.

[213] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[214] Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. What are bayesian neural network posteriors really like? In *International Conference on Machine Learning*, pages 4629–4640. PMLR, 2021.

[215] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

[216] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[217] Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *arXiv preprint arXiv:2309.02390*, 2023.

[218] Mohammad Pezeshki, Amartya Mitra, Yoshua Bengio, and Guillaume Lajoie. Multi-scale feature learning dynamics: Insights for double descent. *arXiv preprint arXiv:2112.03215*, 2021.

[219] Yao Zhang, Andrew M Saxe, Madhu S Advani, and Alpha A Lee. Energy–entropy competition and the effectiveness of stochastic gradient descent in machine learning. *Molecular Physics*, 116(21-22):3214–3223, 2018.

[220] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 18(134):1–35, 2017.

[221] Nicolas Brosse, Alain Durmus, and Eric Moulines. The promises and pitfalls of stochastic gradient langevin dynamics. In *Advances in Neural Information Processing Systems*, pages 8268–8278, 2018.

[222] Omry Cohen, Or Malka, and Zohar Ringel. Learning curves for deep neural networks: a gaussian field theory perspective. *arXiv preprint arXiv:1906.05301*, 2019.

[223] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian convolutional neural networks with many channels are Gaussian Processes. *arXiv preprint arXiv:1810.05148*, 2018.

[224] Jaehoon Lee, Samuel Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *Advances in Neural Information Processing Systems*, 33: 15156–15172, 2020.

[225] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016. URL http://arxiv.org/abs/1609.04836.

[226] Jürgen Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

[227] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

[228] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[229] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[230] Stanislaw Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos J Storkey. Finding flatter minima with sgd. In *ICLR (Workshop)*, 2018.

[231] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. *arXiv preprint arXiv:1806.08734*, 2018.

[232] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Chu Hong Hoi, et al. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *Advances in Neural Information Processing Systems*, 33:21285–21296, 2020.

[233] Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

[234] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International conference on learning representations*, 2018.

[235] Thomas Fel, Louis Bethune, Andrew K Lampinen, Thomas Serre, and Katherine Hermann. Understanding visual feature reliance through the lens of complexity. *Advances in Neural Information Processing Systems*, 37:69888–69924, 2024.

[236] Micah Goldblum, Marc Finzi, Keefer Rowan, and Andrew Gordon Wilson. The no free lunch theorem, kolmogorov complexity, and the role of inductive biases in machine learning. *arXiv preprint arXiv:2304.05366*, 2023.

[237] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

[238] Nick Cammarata, Gabriel Goh, Shan Carter, Ludwig Schubert, Michael Petrov, and Chris Olah. Curve detectors. *Distill*, 2020. doi: 10.23915/distill.00024.003. https://distill.pub/2020/circuits/curve-detectors.

[239] Matthew Bozoukov. Uncovering branch specialization in inceptionv1 using k sparse autoencoders. *arXiv preprint arXiv:2504.11489*, 2025.

[240] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[241] Jie Ren, Qipeng Guo, Hang Yan, Dongrui Liu, Quanshi Zhang, Xipeng Qiu, and Dahua Lin. Identifying semantic induction heads to understand in-context learning. *arXiv preprint arXiv:2402.13055*, 2024.

[242] Chenyang Li, Yitong Liang, Zeyu Shi, Zhi Song, and Ting Zhou. Fourier circuits in neural networks and transformers: A case study of modular arithmetic with multiple inputs. *arXiv preprint arXiv:2402.09469*, 2024.

[243] Sean Ridout, Ilya Nemenman, Ard Louis, Chris Mingard, Radosław Grabarczyk, Kamaludin Dingle, Guillermo Valle Pérez, and Charles London. Bounds on learning with power-law priors. In *APS March Meeting Abstracts*, volume 2024, pages T28–006, 2024.

[244] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

[245] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on information theory*, 22(1):75–81, 1976.

[246] Eric Blais and Li-Yang Tan. Approximating boolean functions with depth-2 circuits. *SIAM Journal on Computing*, 44(6):1583–1600, 2015.

[247] Alexander G de G Matthews, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Sample-then-optimize posterior sampling for bayesian linear models. In *NeurIPS Workshop on Advances in Approximate Bayesian Inference*, 2017.

[248] David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, and Aaron Courville. Deep nets don't learn via memorization. *International Conference on Learning Representations Workshop*, 2017. URL https://openreview.net/forum?id=HJC2SzZCW.

[249] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. Mean-field behaviour of neural tangent kernel for deep neural networks. *arXiv preprint arXiv:1905.13654*, 2019.

[250] Maxwell Nye and Andrew Saxe. Are efficient deep representations learnable? *arXiv preprint arXiv:1807.06399*, 2018.

[251] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[252] Y Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[253] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[254] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[255] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[256] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

[257] N Lakshmipathi. Imdb movie review dataset. URL https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.

[258] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98, 1998.

[259] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[260] Hugo Touchette. The large deviation approach to statistical mechanics. *Physics Reports*, 478(1):1–69, 2009. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2009.05.002. URL https://www.sciencedirect.com/science/article/pii/S0370157309001410.

[261] Thierry Mora and William Bialek. Are biological systems poised at criticality? *Journal of Statistical Physics*, 144(2):268–302, 2011.

[262] Mikio L Braun. Accurate error bounds for the eigenvalues of the kernel matrix. *The Journal of Machine Learning Research*, 7:2303–2328, 2006.

[263] G. P. STECK. Orthant probabilities for the equicorrelated multivariate normal distribution. *Biometrika*, 49(3-4):433–445, 12 1962. ISSN 0006-3444. doi: 10.1093/biomet/49.3-4.433. URL https://doi.org/10.1093/biomet/49.3-4.433.

[264] Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=H1W1UN9gg.

[265] Vijay Balasubramanian. Statistical inference, occam's razor, and statistical mechanics on the space of probability distributions. *Neural computation*, 9(2): 349–368, 1997.

[266] Christopher TH Baker and RL Taylor. The numerical treatment of integral equations. *Journal of Applied Mechanics*, 46(4):969, 1979.

[267] Ariel Caticha. The basics of information geometry. *arXiv preprint arXiv:1412.5633*, 2014.

[268] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6): 84–90, 2017.

[269] Fabio Graetz. How to visualize convolutional features in 40 lines of code, Jan 2019. URL https://towardsdatascience.com/how-to-visualize-convolutional-features-in-40-lines-of-code-70b7d87b0030.

[270] Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Pathological spectra of the fisher information metric and its variants in deep neural networks. *Neural Computation*, 33(8):2274–2307, 2021.

[271] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[272] Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. *Advances in neural information processing systems*, 32, 2019.

[273] Maximus Mutschler and Andreas Zell. Parabolic approximation line search for dnns. *Advances in neural information processing systems*, 33:5405–5416, 2020.

[274] Michal Rolinek and Georg Martius. L4: Practical loss-based stepsize adaptation for deep learning. *Advances in neural information processing systems*, 31, 2018.

[275] Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. Gradient descent: The ultimate optimizer. *Advances in Neural Information Processing Systems*, 35:8214–8225, 2022.

[276] Aaron Defazio and Konstantin Mishchenko. Learning-rate-free learning by d-adaptation. In *International Conference on Machine Learning*, pages 7449–7479. PMLR, 2023.

[277] Maor Ivgi, Oliver Hinder, and Yair Carmon. Dog is sgd's best friend: A parameter-free dynamic step size schedule. In *International Conference on Machine Learning*, pages 14465–14499. PMLR, 2023.

[278] Nicolas Loizou, Sharan Vaswani, Issam Hadj Laradji, and Simon Lacoste-Julien. Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence. In *International Conference on Artificial Intelligence and Statistics*, pages 1306–1314. PMLR, 2021.

[279] Francesco Orabona and Tatiana Tommasi. Training deep networks without learning rates through coin betting. *Advances in Neural Information Processing Systems*, 30, 2017.

[280] Konstantin Mishchenko and Aaron Defazio. Prodigy: An expeditiously adaptive parameter-free learner. *arXiv preprint arXiv:2306.06101*, 2023.

Appendices

A

Occam's razor, simplicity and generalisation

A.1 Formalising Occam's razor

In this section, we begin by briefly summarising early attempts to formalise the vague principle that simpler explanations are better in Appendix A.1.1. We then explain how Occam's razor has been best formalised with Solomonoff induction in Appendix A.1.2, and the problems with this formalisation in Appendix A.1.3.

A.1.1 Statements of the razor

Versions of Occam-like razors have been around at least as far back as Aristotle: "We may assume the superiority ceteris paribus [other things being equal] of the demonstration which derives from fewer postulates or hypotheses" (Posterior Analytics). "Plurality must never be posited without necessity" is William of Ockham's wording, found in his theological work on the "Sentences of Peter Lombard". Einstein had his own version, found in his Herbert Spencer lecture: *"It can scarcely be denied that the supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience."* This statement is often

simplified to *"Everything should be kept as simple as possible, but not simpler"* which is more pithy, although there is no evidence that Einstein ever said this exact phrase.

Blumer [160] showed that a polynomial learning algorithm, as defined by Valiant [161], is obtained whenever there exists a polynomial-time method of producing, for any sequence of observations, a nearly minimum hypothesis that is consistent with these observations. The link between this proof and Occam's razor was disputed in [162] – where Domingos points out that this nearly minimum hypothesis might be highly complex, even if it is found in a small set of models. Domingos claims that the only relation between this result and Occam's razor is: "provided by the infomation-theoretic notion that, if a set of models is small, its members can be easily distinguished by short codes." Domingos makes a distinction between two justifications of the razor

- **OR1** Given two models with the same *generalisation error*, the simpler one should be preferred because simplicity is desirable in itself.
- **OR2** Given two models with the same *training-set error*, the simpler one should be preferred because it is likely to have lower *generalisation error*.

He argues that the first razor is more justified by evidence than the second, as simplicity should be preferred as simpler models are easier to understand, remember and typically less computationally expensive. However, these discussions are incomplete without Solomonoff induction [92], where Occam's razor is formulated in terms of computation.

A.1.2 The coding theorem and Occam's razor

One of the problems many early philosophers found with formalising the razor was exactly what counted as simplicity (see e.g. [163, 164]). The concept underlying Kolmogorov complexity can be traced back to a theorem from Solomonoff [165] in his 1960 paper about general inductive inference, and was independently discovered by Kolmgorov in 1965 [166], and by Chaitin in 1968 [25].

The Kolmogorov complexity of a string x is defined as the length l of the shortest computer program q run on a Universal Turing Machine U that produces x:

$$C_U(x) = \min_q \{l(q) : U(q) = x\}. \tag{A.1}$$

The invariance theorem states that the

$$|C_U(x) - C_V(x)| \leq c \tag{A.2}$$

for some $c(U, V)$ that is independent from x. The Kolmogorov complexity of most strings x is extremely close to their length [167]. Intuitively, this occurs because, for the 2^n strings of length n, there are $2^n - 1$ strings of length less than n. Thus, there is an (almost) 1-1 correspondence between strings of length n and strings (and thus possible programs) shorter than n. Therefore, half the strings in $\{0,1\}^n$ must map to a string in $\{0,1\}^{n-1}$; one quarter into strings in $\{0,1\}^{n-2}$ and so on.

The universal probability [27] of a string x is defined for a prefix UTM U (a prefix UTM is a UTM where no program is the prefix of any other program; required so the sum converges[1]). Prefix Kolmogorov complexity $K(x)$ is related to plain Kolmogorov complexity $C(x)$ via $K(x) = C(x) + C(C(x)) + \mathcal{O}(C(C(C(x))))$ and $C(x) = K(x) - K(K(x)) + \mathcal{O}(K(K(K(x))))$ [168]. Henceforth, we only use prefix Kolmogorov complexity and refer to it as Kolmogorov complexity. Using (prefix) Kolmogorov complexity, the Universal Probability is defined as

$$P_U(x) = \sum_{q:U(q)=x} 2^{-l(q)} / \sum_{q'} 2^{-l(q')}, \tag{A.3}$$

where q' is any program that halts. This is the probability of outputs x when programs q are generated randomly (i.e. appending 0 or 1 to the end of the program

[1] A prefix Turing machine can be constructed as a Turing machine with one unidirectional (where the head can only move from left to right) input tape, one unidirectional output tape, and a bidirectional (the head can only move in either direction) work tape. Input tapes are read only, output tapes are write only. All tapes are binary (no blank symbol), and work tapes initially filled with zeros. For a program to be valid, the prefix UTM must halt with the input head at some point on the input tape. As the input head can only move left to right, it can never access further symbols on the input tape, and thus no valid program can be the prefix of any other valid program. [28]

with equal probability until the program runs). The denominator is guaranteed to be ≤ 1 by the Kraft inequality. Kolmogorov complexity is uncomputable.

The relation between $P_U(x)$ and Kolmogorov Complexity was established in 1974 by Levin [26]

$$2^{-K_U(x)} \leq P_U(x) \leq 2^{-K_U(x)+d} \tag{A.4}$$

where d is independent of x but dependent on U. As a result, we will sometimes replace d with $O(1)$.

Solomonoff induction uses Bayes' rule with a 0-1 likelihood and the universal probability from Eq. (A.3) as the prior to predict the next characters in a binary string. Explicitly, given the a string x, the distribution over continuations y is

$$P(y \mid x) \propto P_U(xy) \tag{A.5}$$

where the xy denotes the concatenation of string x with string y. The posterior distribution minimises expected surprise if the source of the strings is algorithmic in nature [27], and thus the string with the shortest description length is most likely to generalise well. Solomonoff induction is also uncomputable, but has formed the backbone of other theoretical learning agents. See also Section 2.3.

Marcus Hutter's AIXI [169] is a theoretical reinforcement learning agent which combines Solomonoff induction with sequential decision theory. It is Pareto-optimal (there is no other agent that performs at least as well as AIXI in all environments while performing strictly better in at least one environment), and self-optimising (the performance of the policy p will approach the theoretical maximum for the environment μ when the length of the agent's lifetime goes to infinity, when such a policy exists) [170]. AIXI algorithms are also uncomputable, although there are computable approximations like AIXItl, which relies on cutting down the space of allowed programs by setting maximum run-times (avoiding the halting problem). Other related concepts include the minimum-message length formulation [171], where the complexity measure is the complexity of the model plus the amount of information required to encode any discrepancies between the model and the data.

Schmidhuber's speed prior is "a more plausible measure [of simplicity] derived from the fastest way of computing data" [172]. Schmidhuber pointed out that some programs that are simple may be hard to compute (have a long runtime) which does not match entirely with our intuitions of simplicity. The complexity measure associated with this speed prior $K_t(x_n) = \min_q\{l(q) + \log t(q, x_n)\}$ where x_n is a length-n bit string, and a prefix program q computed x_n in t timesteps.

A.1.3 No free lunch, even for Solomonoff Induction

The no free lunch theorem [173] states that no learning agent can outperform any other assuming a uniform distribution over target functions. Solomonoff induction will on average do no better than random if each bit of the target y is sampled at random – it will only generalise well if the target is generated by an algorithmic process.

Lattimore and Hutter [174] argued that Solomonoff induction is a sufficiently good justification of Occam's razor, and that a "free lunch" exists using the universal prior, for all strings of interest (i.e. non-random strings) – as these will be generated by an algorithmic process.

However, this has been met with some pushback from Sterkenburg [175] who pointed out that all results relying on Solomonoff induction must deal with $\mathcal{O}(1)$ terms – specifically, that any string can be made to be arbitrarily simple by some apt choice of Turing Machine. Although these choices become irrelevant in the limit of infinite data streams, the arguments in [174] reduce to the assumption of effectiveness (broadly speaking, computability), which he claims is not a complete justification of Occam's razor. Finally, [176] shows that the convergence of K_U to K_V in the limit of infinite string length does not hold for computable approximations for K_U and K_V. As Solomonoff induction is uncomputable, these approximations would be key for any practical implementation, and the author argues that because these computable approximations would be dependent on the computer used, weakens the justification Solomonoff induction makes for Occam's razor.

A.2 The Case for Simplicity in Practice

The previous section explored theoretical formalisations of Occam's razor, culminating in Solomonoff induction, which suggests a universal prior favouring simpler, algorithmically compressible explanations. While these ideas are powerful, they are uncomputable and abstract. This section grounds the discussion by examining empirical evidence from deep learning, showing that data models are trained on is often simple in nature, and that the models themselves appear to possess an inductive bias towards finding these simple solutions.

Frankle et al. [177] showed that subnetworks with 10% of the parameters of the main network can be trained to achieve parity with the main network (when trained on e.g. CIFAR10), implying that the target function for CIFAR10 is likely to be much simpler than the most complex function neural networks used for the task can represent. Lin et al. [73] argue that deep learning works well because the laws of physics typically select for function classes that are "mathematically simple" (using arguments from statistical mechanics) and so easy to learn. Several studies have attempted to directly calculate the complexity of commonly used image datasets. [89] studied the way in which neural networks learn low dimensional data embedded in a manifold of much higher dimension. Spigler et al. [178] argued that real-world datasets had this property, by calculating an effective dimension $d_{eff} \approx 15$ for MNIST (much lower than the $28^2 = 784$ dimensional manifold in which the data is embedded) and $d_{eff} \approx 35 << 3072$ for CIFAR10. For MNIST, individual numbers can have effective dimensions that are even lower, ranging from 7 to 13 [179]. Bordelon et al. [96] showed that the NTK learns spherical harmonics with larger eigenvalues faster (requiring less data) than smaller ones, which correspond to simpler eigenmodes being leaned faster. An implicit bias towards simplicity may therefore improve generalisation for structured data (but by the no free lunch theorem, if this is the case, it will marginally hurt performance on unstructured data).

A.3 Sources of simplicity bias: the parameter-function map

Having established that real-world data is often simple and that neural networks seem to favour simple solutions, a natural question arises: what is the source of this bias? This section explores one prominent explanation rooted in Algorithmic Information Theory (AIT). By analysing the properties of the mapping from a network's parameters to the functions it can compute, this framework provides theoretical bounds showing that functions with low Kolmogorov complexity are exponentially more likely to be generated by a random choice of parameters.

Dingle et al. [167] showed a link between AIT and generic parameterised input-output maps, and Valle-Perez et al. [64] applied the same techniques to neural networks. This appendix summarises the main points in those papers.

Upper bound Dingle et al. [167] applied the coding theorem [26] to generic input-output maps $M : A \to B$, where $b = M(a)$, to show that the following bound would hold, and under certain mild conditions, be tight

$$P(b) \leq 2^{-K(b)+\mathcal{O}(1)} \tag{A.6}$$

where $K(b)$ is the Kolmogorov complexity of b, and $P(b)$ is the probability that M expresses b upon random sampling of inputs a. Valle-Perez et al. [64] applied this to the parameter-function map of neural networks, defined as

Definition 25 (Parameter-function map)**.** *Consider a parameterised supervised model, and let the input space be $\mathcal{X}$ and the output space be $\mathcal{Y}$. The space of functions the model can express is a set $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$. If the model has p parameters, taking values within a set $\Theta \subseteq \mathbb{R}^p$, then the parameter-function map $\mathcal{M}$ is defined by*

$$\mathcal{M} : \Theta \to \mathcal{F}$$

$$\theta \mapsto f_\theta$$

where f_θ is the function corresponding to parameters $\theta \in \Theta$.

The function space $\mathcal{F}$ of a DNN $\mathcal{N}(\Theta)$ could in principle be considered to be the entire space of functions that $\mathcal{N}(\Theta)$ can express on the input vector space $\mathcal{X}$, but it could also be taken to be the set of partial functions $\mathcal{N}(\Theta)$ can express on some discrete subset $S_X \in \mathcal{X}$. For example, $\mathcal{F}$ could be taken to be the set of possible classifications of images in MNIST. Applying the upper bound to the parameter-function map gives

$$P(f_S) \leq 2^{-K(f_S)+\mathcal{O}(1)}. \tag{A.7}$$

when randomly sampling network parameters θ. Explicitly, given a dataset, the probability of randomly sampling parameters and obtaining f_S is upper bounded by Eq. (A.7). Valle-Pérez et al. [64] used the Boolean datasets with $LZ(f)$ as an approximate upper bound for $K(f)$ as we do here. For the upper bound to be tight, Dingle et al. [65] proposed that the map should be simple, non-linear, non-chaotic and highly redundant.

Lower bound Given a model with p parameters, and a function f, consider the set of parameters that express f, $\Theta_f = M^{-1}(f)$, with elements θ_f. Dingle et al. [66] derived a lower bound for a parameterised function with p parameters

$$P(x) \geq 2^{-K(x|M,p)-\left[p-K(\theta_f|M,p)\right]+O(1)}. \tag{A.8}$$

This bound is tightest when θ_f is chosen to maximise $K(\theta_f|M,p)$. The key term is $[p - K(\theta_f|M,p)]$ – the randomness deficit – or how much simpler θ_f than the typical set of parameters θ (which would be close to maximum complexity, requring p bits to specify). This means the following: if the only way to express a function is with a very simple set of parameters, $P(f)$ can lie far from the upper bound $2^{-K(f)}$ – a "low-K low-P" function. An example of this is l-parity in Section 3.2.2.

See Appendix B.1 for more details on the practical applications of this bound with approximations to $K(f)$.

A.4 Sources of simplicity bias: Double Descent

While the parameter-function map provides a general argument for a bias towards simplicity in highly overparameterised systems, we can make more concrete predictions with linear models (see Section 2.4). We introduce the bias-variance tradeoff, and briefly discuss subtleties around its form for classification compared to regression. We then explain how it arises in linear models, and how double descent means as the number of parameters increases, generalisation improves.

A.4.1 The bias variance tradeoff

The bias-variance tradeoff for regression with mean square error loss is most easily seen form this decomposition of mse loss [180]

$$\underbrace{E_{\mathbf{x},y,D}\left[(f_D(\mathbf{x})-y)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(f_D(\mathbf{x})-\bar{f}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x},y}\left[(\bar{y}(\mathbf{x})-y)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{f}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2},$$

where D is the dataset, f_D is the hypothesis found at the end of training loss minimisation from D (assuming a unique solution for each D), $\bar{f}$ is the mean prediction averaging over datasets, and $\bar{y}$ is the mean label for datapoint x. We can interpret these terms as the variance, noise and bias terms – how much variation over predictions for x different datasets give, noise is irreducible error and how much the mean prediction $\bar{f}$ diverges from the mean label $\bar{y}$ respectively.

However, as pointed out by Domingos [181], this decomposition does not work well for 0-1 loss. He proposes the following breakdown, where y^* is the best prediction, y the true labels and where $f_D = \arg\min_{y'} E_D\left[L(y', y^*)\right]$ is the expected training set prediction (mean for mse loss; mode for 0-1 loss)

$$E_{D,y}\left[L(f_D, y)\right] = c_1 \underbrace{E_y[L(y, y^*)]}_{Noise} + \underbrace{L(y^*, f_D)}_{bias} + c_2 \underbrace{E_D\left[L(f_D, y)\right]}_{variance}. \tag{A.9}$$

With these definitions:

- Noise is the unavoidable component of the loss (independent of the learning algorithm)
- Variance is the average loss incurred by predictions relative to the main prediction (the source of which could be randomness introduced by the training data or stochasticity of the model)
- Bias is loss incurred by the main (mean or mode depending on loss) prediction relative to the optimal prediction.

Note that the possibility of variance being introduced due to stochasticity in the model is not often discussed, as typical examples focus on models where there is one single minimum (typically the case with underparameterised systems like polynomial fitters). However, models like the random learner with LZ complexity cutoff (Appendix E.3.1) are not always fully expressive, but may be able to fit multiple functions with the same test loss, which are picked with uniform probability. This causes variance due to internal model stochasticity.

For mse loss, this equation is the same as the classic bias variance decomposition when $c_1 = c_2 = 1$, $L(a, b) = (a - b)^2$, $y^* = E_y[y]$ and $f_D = E_D[f_D]$. For binary classification with 0-1 loss, this equation correctly reproduces the error when $c_1 = 2P_D(y = y^*) - 1$ and $c_2 = 1$ if $f_D = y$ else -1. $P_D(y = y^*)$ is the probability over training sets in D that the learner predicts the optimal class for x. The total error cannot simply be a sum of the bias and variance terms for 0-1 loss for the following intuitive reason: the performance of a maximally incorrectly biased learning agent (one which always arrives at the wrong answers) will be improved by the addition of some variance. This is explored further in [182]. There, the bias is defined, in words, is the "error due to the fact that the model is too simple with respect to the complexity" and the variance is either due to (1) "match[ing] perfectly the learning set and hence also the noise term" or (2) "the learning algorithm has many different models at its disposal and if the learning set size is relatively small, several of them will realize a perfect match of the learning set. As at most one of them is a perfect image of the best model, any other choice by the learning algorithm

will result in suboptimality." These errors due to variance can therefore be due to variation over models due to variations in the training set, or due to stochasticity in the models themselves. The effect of this is the well-described bias variance tradeoff plot: loss vs number of model parameters, for underparameterised learning agents (fewer indepdent parameters than datapoints). When extremely underparameterised (not able to fit functions remotely close to the training data), the test and training losses will be high. Increasing the number of parameters will decrease both the training loss and test loss, with errors due to bias decreasing (as the model is more able to fit the data) and errors due to variance increasing (the model is able to fit a wider range of functions). The test loss will reach a minimum (when the model almost fits the training data but there are not too many candidate models) and then increase due to variance. The training loss continues to decrease until the model is able to fit the data (typically when the model has the same number of parameters as datapoints), while the test loss increases due to variance. [183] showed that when the train error reaches 0 for the first time, the test loss can diverge.

However, upon adding more parameters, the test loss can *decrease* again. Why?

A.4.2 Double Descent

See also Appendix B.2 for a brief statement of the result.

In the following explanation, we use notation and concepts from Section 2.4. When we train a linear model using gradient descent on the squared loss, starting from zero initialisation, $\theta(t = 0) = 0$, the optimiser introduces a crucial *implicit bias*. The parameter updates are always proportional to $\nabla_\theta L \propto X(X^T\theta - y)$. Because the updates lie in the span of the data features X, the final parameters $\hat{\theta}$ will also lie in this span.

In the over-parameterised case, where there is an entire subspace of solutions that achieve zero training loss, gradient descent will converge to the solution within that subspace which has the minimum L_2 norm. This specific solution is often denoted as the minimum norm solution. The tendency to select this solution is key to understanding double descent.

Let's imagine the target labels y are composed of a simple, clean signal y^* and some random noise ϵ. Suppose the signal y^* primarily lies in the direction of the eigenvectors $|e_k\rangle$ of $K = X^T X$ corresponding to large eigenvalues ρ_k. The noise, in contrast, is spread out across all directions, including those eigenvectors associated with very small eigenvalues.

The Under-parameterized Regime ($p < m$) In the under-parameterized (or classical) regime, the system $X^T\theta = y$ is overdetermined, meaning there is generally no exact solution. Gradient descent on the square loss converges to the unique least-squares solution which minimizes $\left\|X^T\theta - y\right\|^2$. Since the solution is unique, the minimum norm property is not a distinguishing factor.

The test error follows the familiar U-shaped curve. As we increase the number of parameters p towards the number of samples m, the model's capacity increases. Initially, this reduces bias and test error. However, as p gets very close to m, the matrix XX^T (of size $p \times p$) that governs the solution becomes ill-conditioned, with some eigenvalues approaching zero. This forces the model to use large parameter values to fit the training data, leading to a high-norm solution that is sensitive to noise. This results in poor generalisation and marks the "peak" of the test error curve at the interpolation threshold ($p \approx m$) [184].

The Interpolation Peak ($p \approx m$) At the interpolation threshold, the model has just enough parameters to fit the training data perfectly. To do so, it must satisfy $\hat{y} = y = y^* + \epsilon$. The kernel matrix K is invertible but will be ill-conditioned, possessing some eigenvalues ρ_k that are extremely close to zero.

The training dynamics show that components are learned at a rate proportional to their eigenvalue (see Appendix G.4 for a derivation):

$$\langle f|e_k\rangle\,(t) = (1 - e^{-\eta\rho_k t})\,\langle y|e_k\rangle\,,$$

where η is learning rate and ρ_k is the k'th eigenvalue. To fit the noise components $\langle\epsilon|e_k\rangle$ associated with near-zero eigenvalues $\rho_k \approx 0$, the model must find a solution with extremely large coefficients in those directions. The resulting minimum-norm

solution must resort to a very large norm to cancel out all training error, leading to a complex, "spiky" function that overfits the noise and generalises poorly. This creates the peak in test error.

The Second Descent ($p \gg m$) In this regime, the system $X^T\theta = y$ is underdetermined, admitting an infinite number of solutions that perfectly fit the training data. Here, the implicit bias of gradient descent towards the minimum norm solution becomes critical. We can illustrate this using the eigendecomposition of the kernel matrix $K = X^T X$. The eigenvalues ρ_k of K dictate the learning speed for different components of the function.

As we increase the number of parameters p far beyond m, the feature representation becomes richer. This has a regularising effect: the eigenvalues ρ_k of the kernel matrix K increase and the spectrum becomes better conditioned. The smallest eigenvalues move significantly away from zero.

This has two beneficial consequences:

1. **Faster Relative Learning of Signal:** The learning dynamics inherently prioritise fitting the components of the function corresponding to large eigenvalues. Since the signal y^* is associated with these large ρ_k, the model quickly learns the underlying structure of the data. The noise components, associated with smaller (but now not near-zero) eigenvalues, are learned much more slowly. This dynamic separation acts as a form of implicit regularisation.

2. **A "Smoother" Interpolating Solution:** Because the kernel matrix is now well-conditioned, the minimum norm solution that perfectly interpolates y no longer requires an exploding norm. It can fit all the data points, including noise, with a "smoother" function that has a smaller norm. This solution generalises better because it is not forced to make wild excursions to fit noisy data points.

Thus, in the highly over-parameterised regime, the combination of gradient descent's implicit bias for a minimum norm solution and the improved conditioning of the kernel matrix leads to a decrease in test error, creating the second descent.

A.5 Measuring Occams' razor with PAC-Bayes

The preceding sections have explored the theoretical basis for Occam's razor and the evidence for a simplicity bias in neural networks. To formally analyse how such biases affect a model's ability to generalise from a finite training set, we can turn to the PAC-Bayesian framework. This section introduces several key PAC-Bayesian bounds, which provide a powerful tool for quantifying the relationship between a prior distribution over hypotheses (which can encode a simplicity bias) and the expected generalisation error of a learning algorithm.

A.5.1 PAC-Bayesian bounds from McAllester [185]

McAllester [185] provides the following two bounds: the first in the realisable case (where 0 training error can be achieved). Let U be the set of all concepts c consistent with a training set S_m, define a prior measure $P(c)$ over all concepts (their number being finite) c. If $c \in U$, we have

$$\forall D, P_{S_m \sim D^m}\left[\epsilon_g(c) \leq \frac{\ln \frac{1}{P(c)} + \ln \frac{1}{\delta}}{m}\right] \geq 1 - \delta \tag{A.10}$$

$$\forall D, P_{S_m \sim D^m}\left[\epsilon_g(U) \leq \frac{\ln \frac{1}{P(U)} + \ln \frac{m^2}{\delta} + 1}{m}\right] \geq 1 - \delta, \tag{A.11}$$

for some $0 < \delta < 1$, $\epsilon_g(c)$ is the generalisation error of c, and $\epsilon_g(U)$ the expected generalisation error (upon selecting one concept c from U with probability proportional to the prior $P(c)$). The bound on $\epsilon_g(c)$ is smaller for concepts with larger $P(c)$.

In the non-realisable case (c does not have to be consistent with S_m), using the same notation, with V being any finite set of concepts c

$$\forall D, P_{S_m \sim D^m}\left[\epsilon_g(c) \leq \epsilon_S(c, S) + \sqrt{\frac{\ln \frac{1}{P(c)} + \ln \frac{1}{\delta}}{2m}}\right] \geq 1 - \delta \tag{A.12}$$

$$\forall D, P_{S_m \sim D^m}\left[\epsilon_g(V) \leq \epsilon_S(V, S) + \sqrt{\frac{\ln \frac{1}{P(V)} + \ln \frac{m^2}{\delta}}{2m}} + \frac{1}{m}\right] \geq 1 - \delta, \tag{A.13}$$

where $\epsilon_S(c, S)$ is the training error for concept c; and $\epsilon_S(V, S)$ is the expected training error for concepts in V, with concepts weighted by the prior $P(c)$.

Where we have a uniform distribution over c, $P(c) = 1/|\mathcal{H}|$, where $\mathcal{H}$ is the concept space (also referred to as the hypothesis space). In this case Eq. (A.10) can be extended to $P(U)$ trivially, as $P(c)$ is uniform, which is the standard PAC Bound [161],

$$\forall D, P_{S_m \sim D^m}\left[\epsilon_g \leq \frac{\ln \frac{1}{|\mathcal{H}|} + \ln \frac{1}{\delta}}{m}\right] \geq 1 - \delta. \tag{A.14}$$

Note that Eq. (A.10) can also be used to upper bound the expected generalisation error

$$\epsilon(U) = \sum_{c \in U} \epsilon(c) \frac{P(c)}{P(U)} \leq \sum_{c \in U} \frac{P(c)}{P(U)} \frac{\ln \frac{1}{P(c)} + \ln \frac{1}{\delta}}{m} = \frac{\sum_{c \in U} -\frac{P(c)}{P(U)} \ln P(c) + \ln \frac{1}{\delta}}{m} \tag{A.15}$$

After adding $\log(m^2)/m$ to this bound, by convexity of log, we have

$$\epsilon(U) \leq \frac{-\ln P(U) + \ln \frac{m^2}{\delta}}{m} \leq \frac{\sum_{c \in U} -\frac{P(c)}{P(U)} \ln P(c) + \ln \frac{m^2}{\delta}}{m},$$

meaning Eq. (A.15) is is weaker than PAC-Bayes, up to $\log(m)/m$ terms.

A.5.2 PAC-Bayesian bounds from Valle-Pérez and Louis [63], Valle-Pérez et al. [64]

We use the PAC-Bayes bound from [64], a variation on Eq. (A.11) (see [185] for more details). This bounds the expected generalisation error (over runs of the training algorithm)

$$\forall D, P_{S_m \sim D^m}\left[\epsilon_g(Q^*) \leq 1 - \exp\left(-\frac{\ln \frac{1}{P(U)} + \ln \frac{2m}{\delta}}{m-1}\right)\right] > 1 - \delta \tag{A.16}$$

where $Q^*(c) = \frac{P(c)}{\sum_{c \in U} P(c)}$ where as before, c is a concept (hypothesis), U is the set of hypotheses consistent with S, and $P(U) = \sum_{c \in U} P(c)$. When the expectation is not taken over runs (i.e. the bound is for any hypothesis h sampled from the learning agent with probability $Q^*(c)$, meaning a further probability $1 - \gamma$ is required as h may not be representative with small probability), we have

$$\forall D, P_{S_m \sim D^m}\left[P_{h \sim Q^*(c)}\left[\epsilon_g(h) \leq 1 - \exp\left(-\frac{\ln \frac{1}{P(U)} + \ln \frac{2m}{\delta\gamma}}{m-1}\right)\right] > 1 - \gamma\right] > 1 - \delta, \tag{A.17}$$

where $0 < \gamma < 1$ is required as it takes into account the uncertainty due to sampling the hypothesis from $Q^*(c)$.

B

A brief review of generalisation in deep neural networks

Over two decades ago, the statistician Leo Breiman posed a set of deceptively simple questions about machine learning models that remain at the heart of modern deep learning theory. He asked

1. Why don't heavily overparameterised neural networks overfit the data?
2. What is the effective number of parameters?
3. Why does backpropagation not head for poor local minima?
4. When should one stop the backpropagation and use the current parameters?

Answering Breiman's questions has driven progress along several distinct, but complementary, theoretical axes. A few key ones are listed below.

The Kernel Limit One of the most successful approaches has been to study DNNs in the infinite-width limit. Foundational work by Neal [186] and later Jacot et al. [45] showed that as network width approaches infinity, its behaviour under training is perfectly described by a linear kernel method (the NNGP or NTK). This simplifies the problem immensely, as generalisation in kernel methods is well-understood. For

instance, work by [98] provides an exact theory of generalisation based on the kernel's eigenspectrum, showing that functions aligned with the top eigenfunctions are learned fastest. This lens reveals that even in this simplified limit, networks possess a strong spectral bias that helps them generalise, despite being hugely expressive. This line of work tackles all four of Brieman's questions, albeit in a simplified limit.

The Bayesian & Statistical Mechanics View A second approach models the training process itself, often treating stochastic gradient descent (SGD) as a physical process. Work by Naveh et al. [50], Seroussi et al. [140] shows that SGD with infinitesimal noise can be seen as sampling from a Bayesian posterior distribution. This framework connects the optimiser to a principled probabilistic framework and, unlike the kernel limit, can account for feature learning at finite widths. Further work from Bordelon and Pehlevan [187], has developed sophisticated mathematical tools to interpolate between the pure kernel regime and the feature-learning regime, giving us a more nuanced understanding of how network representations adapt during training. These works rely on computing the change to the kernel and then admit ideas from the previous paragraph.

The Algorithmic Simplicity View A third perspective argues that the bias for simple solutions is baked into the very architecture of DNNs, independent of training. Drawing on Algorithmic Information Theory (AIT), Valle-Pérez et al. [64] and Delétang et al. [188] have shown that the space of functions expressible by a randomly initialised network is not uniform; it is heavily biased towards functions with low Kolmogorov complexity. In essence, simple functions are exponentially more likely to be generated than complex ones. This provides a fundamental, a priori reason for why networks start with a "preference" for simple explanations of the data.

Remarks While no single theory is complete, these different research directions have given us a powerful set of tools. They reveal that generalisation arises from a confluence of inductive biases stemming from the model's architecture, the training dynamics, and the optimiser's implicit regularisation. This appendix will review

the key insights from each of these perspectives, building from simple theoretical limits towards the more complex reality of how and why deep learning works.

B.1 A preference towards simplicity

The coding theorem [28] from algorithmic information theory (AIT) (Eq. (A.4)) relates output probabilities to their complexities, assuming generation via a UTM, including the Levin upper bound.

The reader may find it useful to read Appendix A.3 for a sketch derivation. Dingle et al. [189] created a Levin-inspired upper bound, for computable input-output maps. This bound predicts (under certain fairly general conditions that the maps must fulfil, such as a simplicity of the input-output map itself relative to the data, non-linear, non-chaotic, and highly overparameterised) that upon randomly sampling the parameters of an input-output map M, the probability $P(f)$ of obtaining output f can be bounded as

$$P(f) \leq 2^{-K(f|M)+\mathcal{O}(1)} \approx 2^{-a\tilde{K}(f)+b} \tag{B.1}$$

where $K(f|M)$ is the Kolmogorov complexity of f given the model M (usually the explicit dependence on M is removed, so we just write $K(f)$), the $\mathcal{O}(1)$ terms do not depend on the outputs (at least asymptotically), $\tilde{K}(f)$ is a suitable approximation to $K(f|M)$ and a and b are parameters that depend on the choice of $\tilde{K}(f)$ and the map, but not on f. This computable bound was empirically shown to work well over a wide range of input-outpt maps, such as an RNA sequence-to-secondary-structure map, a coarse-grained circadian rhythm ODE map, and an L-system for plant morpohology map; each using an appropriate complexity measure. The range of successes gave confidence that it should be widely applicable, at least for maps that satisfy the conditions needed for it to apply. Furthermore, Dingle et al. [66] derived a statistical lower-bound that predicts most of the probability weight will lie relatively close to the bound, which was tested on a wide range of input-output maps, including a perceptron.

Applying the coding theorem to DNNs This bound was first applied to DNNs by Valle-Pérez et al. [64]. The input-output map to which the bound was applied is the map from the network parameters to the function f it produces on inputs $\mathcal{X}$ which was described in Definition 25. Specifically, the bound was empirically demonstrated to apply to the prior over parameters of DNNs in [64], for some sensible initialisation distribution (such as i.i.d. Gaussians), and suitable complexity measure, for a range of different DNNs on the Boolean system, and several results with NNGPs on more complex systems. In [1], exact results showed biases in the priors of very simple neural networks, like perceptrons, that were consistent with the upper bound. Specifically, it was proved that for a perceptron with no bias term with data from the Boolean dataset, upon randomly sampling the parameters (with a distribution satisfying certain weak assumptions), any value of class-imbalance (entropy) was equally likely. Because there are many more functions with high entropy than low entropy, this is a massive bias towards low-entropy functions, compared to a uniform distribution over functions on the Boolean dataset (expressible by a perceptron). Because low LZ complexity $K_{LZ}(f)$ is implied by low entropy, these results imply a bias in the prior $P(f)$ of perceptrons towards simple functions. They also proved that for infinite-width ReLU DNNs, this bias becomes monotonically stronger as the number of layers grows. Furthermore, it was shown that within classes of functions of any given entropy, there was further bias towards functions with low $K_{LZ}(f)$ (for example, in the maximum entropy class, the function $0101\ldots$ was much more common than the mean).

Note that Yang and Salman [81] showed that the simplicity bias starts to disappear when DNNs with erf or tanh activations enter the "chaotic regime" – which happens for weight variances above a certain threshold, as the depth grows [79] (note that ReLU networks don't have such a chaotic regime in the same way). While these hyperparameters are not typically used for training DNNs in practice, they do show that there exist regimes where there is no simplicity bias (even though the upper bound from [64] still holds trivially). Chapter 4 studies this in more detail.

Algorithmic Information Theory and Universal Prediction. Algorithmic Information Theory (AIT) provides a theoretical foundation for understanding universal prediction through concepts such as Solomonoff Induction and Kolmogorov complexity, and Hutter et al. argue these have significant implications for modern machine learning in [190]. For instance, the inherent limits of different architectures have been empirically mapped to the Chomsky hierarchy of formal languages. An extensive study by Delétang et al. [191] demonstrated that while LSTMs can handle regular and counter-language tasks, and RNNs and Transformers are limited to regular tasks, only architectures augmented with external memory can generalise to context-free and context-sensitive tasks, regardless of training data volume. This suggests that architectural design imposes fundamental constraints on out-of-distribution generalisation. Further connecting theory to practice, the deep link between prediction and compression, long-established in AIT, is highly relevant for large-scale models. Delétang et al. [94] advocate for viewing prediction through the lens of compression, showing that large language models act as powerful, general-purpose compressors that can outperform domain-specific algorithms on varied data modalities like images and audio, and that this equivalence allows any compressor to be repurposed for conditional generation. Building on the aspiration of creating universal predictors, recent work has explored amortising the principles of Solomonoff Induction directly into neural networks.

Grau-Moya et al. [93] demonstrates that by meta-training Transformers and LSTMs on data generated by a Universal Turing Machine, these models can learn universal prediction strategies and acquire reusable algorithmic patterns. However, the source of these powerful inductive biases is not solely a product of training. See also Teney et al. [192], who argue that generalisation capabilities are also deeply rooted in the architecture itself, independent of gradient-based learning. They show that even untrained, random-weight networks possess strong inductive biases and that, contrary to common belief, architectural components can be chosen to bias models towards arbitrary levels of complexity, not necessarily simplicity.

A complementary approach to the Universal Prior-type arguments was taken in [74], where it was shown that upon randomly sampling parameters of ReLUs NN acting on boolean inputs, the functions obtained were on average much less sensitive to inputs than if functions were randomly sampled. Functions with low input sensitivity are simple, thus proving another manifestation of simplicity bias present in these systems.

Inductive bias of different architectures Different architectures, initialisation schemes and activation functions lead to different inductive biases in randomly initialised neural networks [193]. Schoenholz et al. [80] showed a phase transition from ordered to chaotic information propagation in randomly initialised neural networks, depending on the variance of the random Gaussian weights. Teney et al. [192] systematically examined random untrained networks with various activation functions, measuring their complexity in terms of their Fourier spectrum, polynomial expansion and (LZ) compressibility. They showed that standard multilayer perceptrons with ReLU or GELU activations strongly prefer low-frequency functions – effectively smooth input-output mappings – across different depths and weight scales. The opposite is true for Gaussian or sinusoidal activation functions, producing a very different prior in function space, where the inductive bias prefers higher-frequency (generally more complex) functions. See [194] for a similar kernel-based analysis.

B.2 Simplifying limits: Kernel Methods

The most straightforward simplification for analysing deep neural network (DNN) generalisation is to consider the infinite-width limits where the network's behaviour converges to that of a kernel method. This occurs in two primary regimes: the Neural Network Gaussian Process (NNGP) limit and the Neural Tangent Kernel (NTK) limit (see Section 2.6 for the specific initialisation schemes). The foundational work on the NNGP kernel was laid by [186, 48] and further developed by [195], while the NTK kernel was introduced by [45].

In these limits, the DNN becomes a linear model, which simplifies the analysis of its training dynamics considerably. A crucial question then arises: why do these corresponding linear models generalise well, particularly when they are heavily overparameterized?

Kernel methods are linear models on $\varphi(x)$, where generalisation is well understood (described fully in Section 2.4). In the infinite-width limit, the number of effective parameters in the resulting kernel model can be considered to approach infinity ($p \to \infty$), placing it in a highly overparameterized regime. This setting is closely related to the phenomenon of *double descent*, popularised by [196].

While the classic bias-variance tradeoff predicts that test error will increase indefinitely as model complexity (p) surpasses the number of training data points (m), double descent reveals that generalisation error can decrease again in the overparameterized regime ($p > m$). See Appendix A.4 for an in-depth discussion of the bias-variance tradeoff for classification and regression games.

Double descent Belkin et al. [197] investigate the "double descent" risk curve, which describes the test accuracy of some machine learning models as the number of parameters (p) varies. Their work mathematically analyses this curve in two simple data models, using the least squares (with a minimum norm constraint) predictor. They observe that the test risk peaks when p approaches the sample size (m), but decreases as p further increases beyond m.

The central result, Theorem 1, quantifies the prediction risk $\mathbb{E}[(y - x^T\hat{\beta})^2]$ in a Gaussian model, where x is sampled from a standard normal distribution, ϵ is independent standard normal noise, and $y = x^T\beta + \sigma\epsilon$. The risk is given by:

$$\mathbb{E}[(y-x^T\hat{\beta})^2] = \begin{cases} ||\beta_{T^c}||^2 + \sigma^2(1 + \frac{p}{m-p-1}) + ||\beta_T||^2\frac{m}{m-p-1} & \text{if } p \leq m-2 \\ +\infty & \text{if } m-1 \leq p \leq m+1 \\ ||\beta_{T^c}||^2 + \sigma^2(1 + \frac{m}{p-m-1}) + ||\beta_T||^2(1 - \frac{m}{p}) & \text{if } p \geq m+2 \end{cases}$$

Here, $\hat{\beta}_T := X_T^\dagger y$ and $\hat{\beta}_{T^c} := 0$, with $X_T^\dagger$ being the Moore-Penrose pseudoinverse, and T is the set of p selected features. This equation explicitly shows the risk

diverging around $p = n$ and then decreasing in the over-parameterised regime. The intuition behind this is discussed in Appendix A.4.

Eigenspectrum Canatar et al. [98] use statistical mechanics to derive an analytical expression for generalisation error applicable to any kernel and data distribution, where the kernel is trained by gradient descent.

Generalisation error, E_g, is the mean squared error between the estimator $\hat{f}$ and the ground-truth function $\overline{f}(x)$, averaged over the data distribution $p(x)$ and datasets $\mathcal{D}$:

$$E_g = \left\langle \int dx p(x)(\hat{f}(x) - \overline{f}(x))^2 \right\rangle_{\mathcal{D}}$$

Their derived expression for generalisation error relies on the Mercer decomposition of the kernel $K(x, x')$ in terms of orthogonal eigenfunctions $\{\varphi_\rho\}$ and eigenvalues $\{\eta_\rho\}$:

$$\int dx' p\left(x'\right) K\left(x, x'\right) \varphi_\rho\left(x'\right) = \eta_\rho \varphi_\rho(x), \quad \rho = 1, \ldots, N$$

Working with the orthogonal basis set $\psi_\rho(x) \equiv \sqrt{\eta_\rho}\varphi_\rho(x)$, also called a feature map, the target function $\overline{f}(x)$ and the estimator $\hat{f}(x)$ are represented as:

$$\overline{f}(x) = \sum_\rho \overline{w}_\rho \psi_\rho(x) \quad \text{and} \quad \hat{f}(x) = \sum_\rho \hat{w}_\rho \psi_\rho(x)$$

The generalisation error is then given by:

$$E_g = \frac{1}{1-y} \sum_\rho \frac{\eta_\rho}{(\kappa + m\eta_\rho)^2} (\kappa^2 \overline{w}_\rho^2 + \sigma^2 m \eta_\rho)$$

where κ and y are defined by the self-consistent equations:

$$\kappa = \lambda + \sum_\rho \frac{\kappa \eta_\rho}{\kappa + m\eta_\rho} y = \sum_\rho \frac{m\eta_\rho^2}{(\kappa + m\eta_\rho)^2}$$

In these equations, m is the number of training samples, λ is the ridge regularisation parameter, σ^2 is the noise variance, and $\overline{w}_\rho$ are coefficients representing the target function in the eigenfunction basis.

Kernel regression exhibits a strong inductive bias to fit successively higher spectral modes of the target function as the training set size grows. Specifically,

kernel eigenfunctions with larger eigenvalues (η_ρ) can be estimated more rapidly and accurately with fewer samples. This is quantified by the normalized mode error E_ρ, which represents the contribution of each mode to the generalization error and satisfies $\eta_\rho > \eta_{\rho'} \Rightarrow E_\rho < E_{\rho'}$. Consequently, the compatibility between the chosen kernel and the learning task, termed "task-model alignment", is crucial. Tasks where most of the target function's "power" is concentrated in the top kernel eigenfunctions (those with large η_ρ) can be learned more efficiently. See also [96] for generalisation bounds using a similar technique, and simple examples.

Harzli et al. [99] used similar methods for Gaussian Processes (both NNGP kernels and more general kernels) to produce learning curves and demonstrate double descent phenomena in kernel methods. They used width dependent NNGP kernels (while infinite width NNs have fixed kernels, the kernels of finite width NNs vary due to higher order terms, but they can be averaged out) to demonstrate that at the interpolation threshold (where (#datapoints)/(#parameters) $\to 1$) small eigenvalues appear in the data (present due to noise) to which the kernel method overfits, that are not present in the overparameterised or underparameterised regimes ((#datapoints)/(#parameters) $\to 0$ and (#datapoints)/(#parameters) $\to \infty$ respectively).

Spectral bias on the hypersphere and hypercube For fully-connected ReLU networks, there are several works that provide a full description of the NTK eigenbasis. For fully-connected ReLU networks with input drawn from the d-dimensional hypersphere, the NTK is a dot-product kernel. Its eigenfunctions are the spherical harmonics on the hypersphere. The symmetry of the architecture together with the data symmetry leads to a polynomial decay in the eigenfunction k^d with the degree k of the spherical harmonics [198, 199].

This means the kernel deems low-frequency spherical harmonics as "important" functions (high eigenvalue), and high-frequency as "hard" functions (low eigenvalue). Intuitively, this quantifies the simplicity bias of the network in a basis of functions: smooth, low-order functions lie in top-eigenvalue eigenspaces, whereas highly

oscillatory or complex dependences lie in low-eigenvalue eigenspaces. A direct consequence is a spectral bias in learning. When one trains a neural network (in the kernel regime) or does kernel regression on data, the component of the target function along high-eigenvalue eigenfunctions is learned first and with few samples, while components along low-eigenvalue eigenfunctions require many more samples to fit.

Empirical studies have demonstrated this frequency bias: for instance, when fitting a target function on the unit circle that is a mixture of sines and cosines, neural nets quickly learn the low-frequency modes and only later fit the high-frequency components [101]. Rahaman et al. [200] showed that standard deep networks trained on regression tasks exhibit a Fourier frequency bias: the error in fitting different Fourier modes is controlled by the mode frequency, with low-frequency signals learned much faster [98].

These arguments transfer to neural networks trained with data from the hypercube. The NTK eigenfunctions on the hypercube are parity functions on subsets of k input bits, and eigenvalues decrease as k grows. Thus, a network can learn any function that is, say, an XOR of a few bits relatively easily (those correspond to eigenfunctions with high eigenvalue), but learning the parity of all n bits (the hardest, $k = n$ eigenfunction) is extremely slow and sample-inefficient – essentially requiring memorization since that function lies in a low-eigenvalue subspace [101]. This quantitatively explains why neural nets struggle with high-order parity (a well-known "hard" function class in theory) unless aided by exponential data or special architectural tweaks [201]: the inductive bias is simply misaligned with that function. On the other hand, a function like a single-bit identity or a simple conjunction of a few bits has most of its variance in low-order parity components and is learned readily. These insights bridge the gap between the empirical simplicity bias and a theoretical characterisation: networks have an implicit bias toward functions expressible by low-degree polynomials or low-frequency Fourier components, which are precisely the "simple" patterns in the input space. The spectral bias argument can be extended to other architectures like CNNs [202].

B.2.1 Shortcomings of the kernel approximations

There is a rich literature on the sample complexity for kernels which provides a full theory of generalisation for kernels and hence infinitely wide neural networks [203]. However, there are several known datasets where neural networks strongly outperform kernels in terms of their sample complexity [204, 205, 206, 207]. That is, neural networks learn the target function with a much lower number of datapoints. This is generally attributed to feature learning, the ability of the neural network to adapt its hidden representation [208]. This part of deep learning cannot be understood through a kernel perspective.

While it is well known that both the NNGP and NTK limits cannot perform representation learning (see e.g. [42, 209]), it is worth noting that finite-width neural networks inevitably learn features (to some degree or other), whether trained with a Bayesian optimiser or not.

The ability to learn representations is a very important property of DNNs – it allows transfer learning, and a greater degree of representation learning has been linked to simpler functions [210, 3] and better generalisation [130, 8].

B.3 Simplifying limits: Infinitesimal GD

Naveh and others in several works [211, 212, 50] showed that neural networks trained with gradient descent and a small amount of white noise can be described by an over-damped Langevin equation that converges (under some further light conditions) to the Boltzmann distribution. This distribution is of the form $P_B(f|S) \propto e^{S(f)-\beta E(f)}$ [213] and lends itself to a Bayesian interpretation: where the configurational "entropy" $S(f)$ counts the number of states that generate f (in the Bayesian interpretation, the prior), and $E(f)$ the energy (the log likelihood or loss function). The β parameter controls the temperature of the posterior [214], where a large value (low temperature) makes the likelihood dominate (so only maximum likelihood functions have significant weight in the posterior; and are further weighted by the prior), and in the limit of $\beta \to 0$ (the high temperature

limit), the posterior is identical to the prior. For example, in [50], training a neural network with GD, infinitesimal learning rate, white noise and weight decay leads to the following posterior distribution over weights θ

$$P(\theta) \propto \exp\left(-\frac{||\theta||^2}{2\sigma_\theta^2} - \frac{L(S,\theta)}{2\sigma^2}\right) \tag{B.2}$$

where $L(S,\theta)$ is the loss function on some training set S with weights θ and σ and σ_θ are hyperparameters controlled by the magnitude of the white noise and weight decay, respectively. The first term in the brackets can be thought of as the prior over parameters, and the second is the likelihood. In the limit of infinite width, Eq. (B.2) is the NNGP limit [215], but at finite width, feature learning will happen.

Statistical mechanics approach This posterior allows for the study of feature learning. The key insight is that while the posterior over the network's outputs remains approximately a Gaussian Process (GP), its kernel is no longer fixed by the architecture but adapts to the data.

Work by Seroussi et al. [140] formalises this by deriving a set of self-consistency equations that govern the network's behaviour at a finite width. In their model, the pre-activations in each layer remain nearly Gaussian, but their covariance matrices (the kernels) are modified in a top-down manner. For an L-layer network, this results in a data-aware GP described by:

- **Final Output:** The network's average output, $\overline{f}$, is found using a standard GP inference formula, but with a *learned* kernel Q_f that has adapted to the training data:
$$\overline{f} = Q_f[Q_f + \sigma^2 I_n]^{-1} y$$
- **Kernel Self-Consistency:** The kernels of the hidden layers are determined by a system of equations that couple them to both upstream and downstream layers. For instance, the pre-kernel of the penultimate layer, $K^{(L-1)}$, is modified based on the training error:
$$[(K^{(L-1)})^{-1}]_{\mu\nu} = [(Q^{(L-1)})^{-1}]_{\mu\nu} - \frac{1}{N_{L-1}} \mathrm{Tr}\left\{A^{(L)} \frac{\partial Q_f}{\partial [K^{(L-1)}]_{\mu\nu}}\right\}$$

where the term $A^{(L)}$ depends on the discrepancy between the network's predictions and the true labels, $(y - \overline{f})$.

A key aspect of this thermodynamic theory is the emergence of a feature learning scale (FLS): $\chi = N_{L-1}^{-1} \epsilon^\top Q^{(L-1)} \epsilon$, where N_{L-1} is the width of the penultimate layer, ϵ represents the normalized training error $(y - \overline{f})/\sigma^2$, and $Q^{(L-1)}$ is the data-agnostic kernel of that layer. This scale represents the leading term in a Taylor expansion of the learned output kernel, Q_f, in powers of $1/N_{L-1}$. When χ is small, the network remains close to the infinite-width Gaussian Process (GP) regime. However, when χ is of order 1 or larger, the network enters a strong feature learning regime, where the kernel adapts significantly to the data.

A crucial insight is that χ can be large even when the network width N_{L-1} is very large. In common scenarios, the FLS can be approximated as $\chi \approx \lambda \cdot \text{MSE}/\sigma^2 \cdot n/N_{L-1}$, where n is the dataset size, MSE is the mean squared error on the training set, and λ is the scale of the dominant eigenvalues of the initial kernel. Because λ often scales with n (i.e., $\lambda = O(n)$), the FLS can be of order 1 or greater due to the resulting n^2 factor in its scaling, even for very wide networks. This is why simple perturbation theories based on $1/N$ often fail; they incorrectly treat the FLS on the same footing as other $O(1/N)$ terms, whereas this theory treats the FLS non-perturbatively. The theory's ability to capture this scale separation is central to its success in predicting the behaviour of finite-width networks.

See also Bordelon and Pehlevan [187], who develop a path integral formulation of gradient flow dynamics in infinite-width networks, which includes a scalar parameter to allow interpolation between the feature learning and non-feature learning regimes. They also show an equivalence between their formulation and the tensor programs formulation of Yang and Hu [42]. The tensor programs formulation is used to demonstrate that embeddings for a Word2Vec task are learned in the infinite width limite with μP but not NTK parameterisation.

Example: Grokking Aside from using the width and parameterisation to control feature learning, feature learning can also emerge over time. Grokking [216] sometimes occurs when training neural networks, and is characterised by (1) an initial period where the training accuracy increases to near its maximum, while the test error remains poor (2) a very long period where neither accuracy changes significantly and (3) a period where the test accuracy decreases significantly, without the training accuracy changing at all. Varma et al. [217] explained this by a process of delayed feature learning: the model first memorises the data by only learning weights in the last layer (phase 1), weight decay encourages a lower weight norm solution, which involves slowly learning better representations in the first layer (phase 2), and test accuracy rises if the learned representations are good (phase 3). See also [218] for an explanation based on the rate of modes being learned. This does suggest the possibility of a kind of inverse grokking: where the slow-to-learn representations are worse than the initial ones

Shortcomings However, for SGD the equivalent coarse-grained differential equation reduces to Langevin equation with anisotropic noise [212, 219] and doesn't exactly converge to the Bayesian posterior Eq. (B.2) [220, 221]. Furthermore, even though GD with white noise does converge exactly to a Bayesian posterior in the limit of infinite training steps, the rate of convergence can vary between problems. Nevertheless, it has been conjectured that with small step size, SGD may approximate the Bayesian posterior [50, 222, 2].

B.4 Reintroducing SGD

While the infinite-width kernel perspective provides a powerful baseline for understanding generalisation, it does not admit feature learning (Appendix B.2). Even the models that do (Appendix B.3) operate in the continuous gradient descent limit (i.e. no minibatching or finite learning rates). Given the often-repeated intuition that if you tune one parameter, tune learning rate, we need to address the effect of finite learning rate to understand generalisation.

Early work comparing Stochastic Gradient Descent (SGD) trained networks to their kernel counterparts found close, but not perfect, agreement [195, 51, 223, 224]. A direct bridge between the kernel and optimisation-centric views was established by Mingard et al. [2] (see also Section 6.1). They demonstrated that for many standard architectures and datasets, the distribution of functions found by SGD, $P_{\text{SGD}}(f|\mathcal{S})$, closely approximates the Bayesian posterior, $P_{\text{B}}(f|\mathcal{S})$, which in the infinite-width limit corresponds to the NNGP. They termed the generalisation arising from this inherent architectural prior "First-order generalisation." Crucially, however, they showed that optimiser hyperparameters like the learning rate and batch size could systematically shift the SGD solution away from the Bayesian posterior, often correlating with changes in generalisation error. This confirms that the optimiser is not a mere search algorithm but an active source of inductive bias.

One of the most studied mechanisms for this bias relates to the geometry of the loss landscape. It was famously observed that small-batch SGD tends to find solutions that generalise better than those found by large-batch SGD [225]. The hypothesised reason was that the noise inherent in small-batch updates helps the optimiser settle in "flat" minima, which are associated with more robust solutions, as opposed to "sharp" minima [226]. While later work showed that much of this effect can be attributed to the larger number of optimisation steps taken by small-batch methods [227, 228, 229], the connection between optimiser noise and solution geometry remains key. Mechanistically, the anisotropic noise in the SGD update rule is thought to act as an implicit regulariser that pushes iterates towards flatter regions of the loss surface [219, 230].

Beyond the geometry of the final solution, the trajectory of gradient descent is itself biased. A fundamental property is spectral bias: neural networks preferentially learn simple, low-frequency functions first during training [231]. This bias towards simplicity provides a compelling explanation for why overparameterised models do not immediately overfit to high-frequency noise in the training data (which would be a function of the architecture and explainable with methods in Appendices B.1 and B.3). The specific characteristics of the optimiser also play a significant role.

For instance, Zhang et al. [219] derive Langevin-type equations for SGD and GD (see Eq. (B.2)), and argue that the presence of the anisotropic noise in the SGD equation pushes SGD towards flatter minima. This case was also made in [230], and [232] argued that Adam typically generalises less well than SGD because the noise induced by SGD has a heavier tail than induced by Adam. However, such comparisons must be made with care, as with meticulous hyperparameter tuning, the behaviour of adaptive optimisers can converge towards that of SGD with momentum [233].

These distinct biases—towards flat minima and low-frequency functions—are not mutually exclusive and their interplay shapes the final model. Their effects can be observed in complex emergent behaviours, such as the layer-wise alignment of features, which can also be influenced by optimiser hyperparameters [3]. Early alignment of features to the data has been argued to improve performance [210], but it can sometimes hurt generalisation [8]. It is also clear that the solutions found by SGD represent only a subset of possible good solutions. For instance, more expensive Bayesian methods like MCMC can explore the posterior distribution more fully and often achieve superior performance [214], underscoring that SGD's inductive bias is a specific and powerful, but not exhaustive, path to generalisation.

B.5 Mechanistic Interpretability

While the studies discussed in the previous parts of this review treat neural networks mostly as black-box functions, a complementary line of research asks how exactly neural networks internally represent data – in effect, opening the black box to find mechanistic explanations for a network's behaviour. Mechanistic interpretability seeks to reverse-engineer the specific circuits and algorithms encoded in a trained network's weights. The goal is to move beyond coarse measures (like complexity or norm-based capacity) and identify neurons, attention heads, layers, or combinations thereof that correspond to meaningful functions or subroutines within the network.

Inductive bias and real world data Different neural networks have different kinds of inductive biases. How much these biases improve or worsen generalisation depends on the structure of the target function. A common example is training an FCN and a CNN on an image dataset like CIFAR-10: both will perform better than random, but the CNN will reach a significantly higher test accuracy. Convolutional networks trained on images have been found to latch onto simple, low-level cues (like textures or colours) rather than more complex global structures. Geirhos et al. [234] showed that ImageNet-trained CNNs are strongly biased toward texture recognition rather than object shape. When presented with images where texture and shape cause conflict, CNNs predominantly follow the texture. Texture cues are, in a sense "simpler" or more immediate statistical features of images (requiring only local filtering), whereas global shape integration is more complex. This bias towards easy-to-pick-up features can hurt robustness, but it is an instance of the network's preference for a simple explanation of the data (here, classifying by texture) when one exists. See [235] for a further discussion of the complexity of features in CNNs and how they arise during training.

Another line of work has examined transformer architectures for algorithmic or logical tasks. Bhattamishra et al. [71] found that Transformers have a simplicity bias analogous to deep networks: they more readily learn low-sensitivity (sparse) Boolean functions than complex ones. For instance, a Transformer trained on a Boolean function that depends only on a small subset of input bits (with the rest being irrelevant noise) will generalise well, whereas learning a highly "entangled" function like parity (which depends on all bits) is notably difficult without special measures. This suggests the inductive biases of modern sequence models also favour functions with simple structures (e.g. ones that can be decomposed into a few salient features or rules), even if the models in principle have the capacity to implement very complex mappings [235].

Inductive bias only improves the performance of the model when it aligns with the target. Having well-performing models relies on real-world data being highly structured. Indeed, Goldblum et al. [236] argue that this bias is one key

reason we can have "general-purpose" models: real-world tasks themselves produce data that are far from fully random and instead have low underlying complexity, benefiting neural networks, which innately favour such low-complexity patterns. In their experiments, architectures specialised for one domain can often compress or model data from another domain if it shares a low-complexity structure, and even large pretrained language models with random weights preferentially generate low-complexity (compressible) sequences rather than arbitrary complex ones [236]. This surprising observation – that an untrained GPT-style model already favours simplistic output patterns – reinforces that a great deal of inductive bias comes from architecture alone, not just from gradient descent [192].

Bias towards low-complexity functions after training [107] showed that neural networks are expressive enough to fit randomly labelled data. This raised the question of why neural networks trained on non-random labels generalise at all, since they would be perfectly capable of interpolating the training data while having random chance accuracy on the test data. This highlights that, beyond random initialisation, the neural network training process itself introduces an inductive bias. Mingard et al. [2] argue that the posterior distribution over functions retains a simplicity bias – among all functions consistent with the training set, SGD-trained networks tend to land on ones of relatively low complexity. They offer empirical evidence on non-Boolean data that SGD is more likely to learn functions with larger $P(f)$ (see also [50] for a theoretical explanation using infinitesimal GD with weight decay). Kalimeris et al. [75] provided complementary evidence by examining training dynamics: they observed that SGD learns functions of increasing complexity over time, effectively learning a simple, approximately linear decision boundary in the early epochs and then gradually fitting more complex aspects of the target function in later epochs Early in training, almost all of the network's performance can be attributed to a "simple classifier" component, and only with more iterations does the model incorporate higher-order or more complex features. See Appendix B.2 for the kernel explanation of this phenomenon (or see Canatar et al. [98]).

Interpreting CNNs – Circuits and feature visualisation In CNNs, interpretability research has progressed from identifying individual neurons that detect human-interpretable concepts to mapping out multi-neuron interactions, or circuits, that represent higher-level features. Early work used feature visualisation to synthesise an input image that maximally activates a given neuron or layer, revealing the feature that neuron represents (e.g. a neuron might fire for textures like "striped pattern" or specific objects like "dog faces").

In a series of articles (see e.g. [237, 57]), Olah et al. demonstrated that CNNS learn circuits for meaningful visual concepts. One notable example is a "curve detector" circuit [238]: lower-layer neurons detect short curves at various positions; a mid-layer neuron sums these to detect longer curves; that neuron in turn feeds into a higher-layer neuron that detects round objects (like wheels or pupils), together forming a hierarchical circuit for round shapes. Such precise visual explanations illustrate how a network builds complex features (like detecting an animal face) by composing simpler ones (edges, textures, etc.). However, purely correlational methods face the challenge of polysemantic neurons (neurons that activate on multiple features).

Bozoukov [239] used sparse autoencoders on InceptionV1's mixed layers to isolate more interpretable "feature vectors", which enabled tracing connections between features across layers to reconstruct circuits. They uncovered branch-specific circuits in the network – e.g. a chain of features detecting animal faces: early layers detect oriented animal parts (faces facing left or right), a next-layer feature combines them into a generic animal face detector, which then branches into specialised detectors for dog faces and dog legs in later layers. This kind of mechanistic story, where one can follow the activation flow through a sequence of feature detectors, represents a state-of-the-art understanding of how CNNs implement complex visual recognition. It provides a satisfying mechanistic explanation (complete with visual evidence) for tasks that the network learned – essentially reverse-engineering a portion of the network's computation graph in human-understandable terms.

Transformer models – Induction heads and algorithms In Transformers, recent work has identified small-scale circuits inside large models that correspond to specific algorithms the model has learned. A prime example is the discovery of induction heads in Transformers. Induction heads are particular attention heads that implement a simple copy-and-paste algorithm: given a sequence pattern "[A][B] ... [A]", an induction head learns to attend from the second "[A]" back to the token "[B]" that followed the first "[A]", effectively retrieving "B" as the predicted continuation. Olsson et al. [240] provided multiple lines of evidence that induction heads are the mechanistic basis of in-context learning in Transformers. They observed that at a certain training stage, models undergo a sudden jump in their ability to do in-context prediction of sequences (for example, continue a list in the style of the prompt), and this coincides with the emergence of one or two attention heads that reliably implement the above copy mechanism. By ablating those heads, the in-context learning ability drops, confirming a causal role. This finding is remarkable because it isolates a transparent algorithm within the black-box: the model learns to implement a memory lookup and retrieval operation entirely through a couple of attention heads (which are simple matrix-weighted operations). It's a rare case where one can point to specific weights in a large language model and say "this is performing task X via mechanism Y." Subsequent work has extended this analysis to larger models and more complex behaviours. For instance, Ren et al. [241] identified "semantic induction heads" that not only copy tokens but do so in a way that respects word semantics (copying the next word of a repeated sequence even with intervening synonyms), showing the versatility of these circuits.

Beyond induction heads, interpretability researchers have attempted to reverse-engineer entire algorithms learned by Transformers on small tasks. For example, Li et al. [242] fully explained how a tiny Transformer performs modular arithmetic. Nanda et al. [58] studied grokking in transformers performing modular arithmetic, by reverse-engineering the algorithm the network learned for modular addition. They discovered that the 1–layer Transformer had learned to implement addition by internally converting numbers to a discrete Fourier representation (essentially

representing integers as complex phases on a circle) and performing rotations. Accordingly, they defined progress measures for each component of the algorithm (e.g. how well the Fourier conversion sub-circuit was formed) and tracked them during training. They found that training proceeded in three phases: (1) memorization – the model first purely memorizes many training examples, (2) circuit formation – gradually the Fourier addition circuit emerges and gains strength, and (3) cleanup – finally the model prunes away the now-unneeded memorized solutions, relying purely on the general algorithm. What appeared as a sudden "grokking" jump in test accuracy was explained mechanistically as the point when the algorithmic circuit surpassed memorisation in importance.

C

Appendices for Chapter 3

C.1 DFCNs vs FCNs

The DFCN is not a typical discrete approximation of a neural network, as the bias terms are all functions of weights, to make sure each neuron represents a clause. One consequence is that $P(f)$ is strongly width-dependent, unlike standard FCNs on Boolean data [64]. In the limit of infinite width ($\alpha_w \to \infty$), at initialisation, $P(f)$ will be entirely dominated by a constant function (Lemma 13). This is because adding more clauses can only set more inputs to True – eventually, every input will be covered by at least one clause. See Fig. C.3 for an empirical demonstration.

C.2 Why does parity generalise badly?

In Section 3.3.3 we discuss details about training DFCNs on k-parity functions. An interesting result is that for the most complex function (7-parity), the test accuracy gets worse the larger the training set. This can be understood by thinking about what a network will predict on unseen test data. Consider an example for $n = 4$ where the target function is 4-parity: `"0110100110010110"`. Let's assume in the following examples that we train to the minimum norm solution.

1. Suppose that we train on the first four bits ($m = 4$). The minimum norm solution for this is 2-parity (parity on the first two bits). Then, f is just the

first 4 bits repeated, `4x"0110"` = `"0110011001100110"`. On the remaining 12 bits, our accuracy is 33%.

2. Now we train on the first 8 bits. The minimum norm solution is now 3-parity, `2x"01101001"` = `"0110100101101001"`, which has 0% test accuracy.

This argument can be straightforwardly generalised to larger n. Choosing training examples in this way, when trying to learn parity generalisation, will get worse the larger the training set.

C.3 Important differences between $K_{DNF}(f)$ and $K_{LZ}(f)$

One important class of functions where the two measures differ significantly is functions with repeating patterns. Consider the string representation of f.

1. Consider the function $f =$ `"1001"` $\times 2^{n-2}$. This function is 2-sparse, represented by the DNF $(\neg x_1 \wedge x_2)$ (1 clause, 2 literals). As n increases, its DNF complexity remains fixed at 2. It's Lempel-Ziv complexity $K_{LZ}(f) = C + \log_2 n$ (constant term C for encoding the repeating string and the $\log n$ term from the repetitions)

2. However, if we generate a function f by repeating the string `"01001"` (truncating at the end), the result is not a k-sparse function. As a result, its $K_{DNF}(f)$ will not be constant.

See Fig. C.1 for empirical data showing the discrepancy between these measures.

Empirical work in [64] shows that $K_{DNF}(f)$ and $K_{LZ}(f)$ are correlated. However, they have very different maximum complexities. Given that a function can be represented in its string representation with 2^n bits – a really good complexity measure should never go above $2^n + O(1)$. The maximum value of $K_{DNF}(f)$ is $\frac{n}{2}2^n$, or $O(n2^n)$. In contrast, $K_{LZ}(f)$ has a worst-case of $O(2^n)$.

Fig. D.1 shows how the two measures scale for random boolean functions (generated by assigning a 1 or 0 randomly for each input x) and parity, as a function

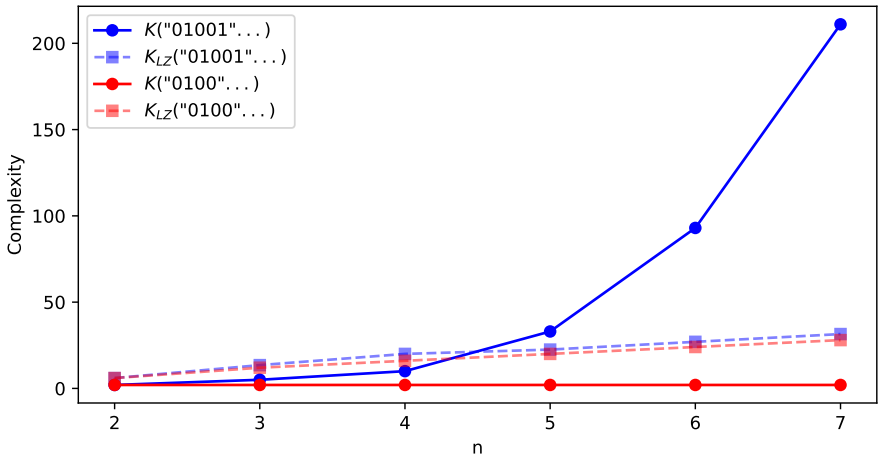


Figure C.1: $K_{DNF}(f)$ **vs.** $K_{LZ}(f)$ **on repeating functions**. **Dashed** lines show $K_{LZ}(f)$ and **solid** lines show $K_{DNF}(f)$. The **red** curves show the 2-sparse function $f = \texttt{"1001"} \times 2^{n-2}$. $K_{DNF}(f)$ remains constant at 2, and $K_{LZ}(f)$ grows slowly with n. By contrast the **blue** curves show the function generated by the repeated pattern `"01001"` which yields a function that is not 2-sparse and therefore does not enjoy a small, constant $K_{DNF}(f)$, but does enjoy a $K_{LZ}(f)$ that grows at a similar rate to the $K_{LZ}(f)$ of the 2-sparse function.

of n. Random functions should have complexities close to 2^n (as they are not compressible beyond their string representation), which is the case for both $K_{LZ}(f)$ and $K_{DNF}(f)$. Parity, on the other hand, scales very differently. Future work could determine the fraction of functions with complexity greater than 2^n.

C.4 Proofs from Mingard et al. [1]

Proof of Proposition 3.1.1

Proof. Fix an input dimension n. Let $f : \{0,1\}^n \to \{0,1\}$ be an arbitrary Boolean function and set

$$t \;=\; \Big|\{\,\mathbf{v} \in \{0,1\}^n \;:\; f(\mathbf{v}) = 1\}\Big|. \tag{C.1}$$

We adopt the notation for a clause C as laid out in Definition 3.

Define the set of all DNFs by $\mathcal{F}_{\text{DNF}}$ and an equivalence relation R_{DNF} given by permutations of DNF clauses, such that the set of equivalence classes is $\mathcal{F}_{\text{DNF}}/R_{\text{DNF}}$. Similarly, define the set of all DFCNs of width 2^{n-1} by $\mathcal{F}_{\text{DFCN}}$ and an equivalence relation R_{DFCN} given by permutations of rows in $W^{(1)}$, such that the set of

equivalence classes is $\mathcal{F}_{\text{DFCN}}/R_{\text{DFCN}}$. We now show that there exists a bijective map $\mathcal{G} : \mathcal{F}_{\text{DNF}}/R_{\text{DNF}} \to \mathcal{F}_{\text{DFCN}}/R_{\text{DFCN}}$.

$\mathcal{G}$ is injective We begin by assuming that $t \leq 2^{n-1}$. Define a depth-two DFCN with layer sizes $\langle n, t, 1\rangle$ and $\beta = 1$ (following Definition 6). For $i \in \{1, ..., t\}$, let $\mathbf{v}^{(i)}$ be the i-th input vector for which f outputs True such that $\gamma_i = \sum_j v_j^{(i)}$ is the number of positive literals in clause C_i. Set

$$W_{ij}^{(1)} = \begin{cases} +1 & \text{if } v_j^{(i)} = 1, \\ -1 & \text{if } v_j^{(i)} = 0, \end{cases} \qquad b_i^{(1)} = 1 - \gamma_i, \tag{C.2}$$

$$W_i^{(2)} = 1, \qquad b^{(2)} = 0. \tag{C.3}$$

For any $\mathbf{v} \in \{0, 1\}^n$,

$$z_i(\mathbf{v}) = \sum_i W_{ij}^{(1)} v_j + b_i^{(1)} \;=\; \gamma_i - d(\mathbf{v}, \mathbf{v}^{(i)}) + (1 - \gamma_i) \;=\; 1 - d(\mathbf{v}, \mathbf{v}^{(i)}), \tag{C.4}$$

where $d(\cdot, \cdot)$ denotes the hamming distance. $z_i(\mathbf{v}) = 1$ iff every literal in C_i is satisfied and $z_i(\mathbf{v}) \leq 0$ otherwise. Since $\sigma(z_i) = \text{ReLU}(z_i) = \max(z_i, 0)$,

$$\sigma\Big(z_i(\mathbf{v})\Big) = C_i(\mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{v}^{(i)} = \mathbf{v}, \\ 0 & \text{if } \mathbf{v}^{(i)} \neq \mathbf{v}. \end{cases} \tag{C.5}$$

In other words, the output of the first layer is 1 if the clause C_i is satisfied and 0 not. $W^{(2)}$ then effectively acts as an OR operator, giving us

$$f_\theta(\mathbf{v}) = \mathbb{1}[W^{(2)}\, \sigma\Big(W^{(1)}\mathbf{v} + b1\Big) + b2 > 0] \tag{C.6}$$

$$= C_1(\mathbf{v}) \vee \cdots \vee C_t(\mathbf{v}) \;=\; \Phi_f(\mathbf{v}) \;=\; f(\mathbf{v}). \tag{C.7}$$

If $t > 2^{n-1}$, we instead use a network with layer sizes $\langle n, 2^{n-1} - t, 1\rangle$ and let $\mathbf{v}^{(1)}$ be the i-th input vector for which f outputs False (which must be $\leq 2^{n-1}$). We then set $\beta = -1$, which negates $W_i^{(2)}$ and sets $b2 = 1$, giving us the following parameters,

$$W_{ij}^{(1)} = \begin{cases} +1 & \text{if } v_j^{(i)} = 1, \\ -1 & \text{if } v_j^{(i)} = 0, \end{cases} \qquad b_i^{(1)} = 1 - \gamma_i, \tag{C.8}$$

$$W_i^{(2)} = -1, \qquad b^{(2)} = 1. \tag{C.9}$$

z_i still outputs 1 if the clause C_i is satisfied and 0 if not, but $W^{(2)}\mathbf{z} < 0$ if any of the False clauses are satisfied. Thus, the only way to obtain a positive output inside the indicator function in Eq. (C.6) is if all C_i are not satisfied (since $b2 = 1$ brings the value above 0 in this case). This then gives us

$$f_\theta(\mathbf{v}) = \mathbb{1}[W^{(2)}\,\sigma\Big(W^{(1)}\mathbf{v} + b1\Big) + b2 > 0] \tag{C.10}$$

$$= \neg\,[C_1(\mathbf{v}) \vee \cdots \vee C_t(\mathbf{v})] \;=\; \Phi_f(\mathbf{v}) \;=\; f(\mathbf{v}). \tag{C.11}$$

We can pad the width of the network to be 2^{n-1} by adding rows of zeros to $W^{(1)}$ and setting the rest of the weights according to Definition 6. We have extra degrees of freedom in the permutations of these rows and thus invoke the equivalence relation R_{DFCN}, proving injectivity of $\mathcal{G}$.

$\mathcal{G}$ is surjective Conversely, let a parameter tensor $\theta = (W^{(1)}, b1, W^{(2)}, b2, \beta)$ satisfy the constraints of Table 3.1 with hidden width 2^{n-1}. Keep only the indices i for which $W_i^{(2)} = \beta$; there are $t \leq 2^{n-1}$ of them. For each such i, define the clause

$$C_i = \Big(\bigwedge_{\{j:W_{ij}^{(1)}=+1\}} x_j\Big) \wedge \Big(\bigwedge_{\{j:W_{ij}^{(1)}=-1\}} \neg x_j\Big). \tag{C.12}$$

Following the same reasoning as before, we conclude that

$$f_\theta(\mathbf{v}) = \mathbb{1}[W^{(2)}\,\sigma\Big(W^{(1)}\mathbf{v} + b1\Big) + b2 > 0] = \begin{cases} C_1(\mathbf{v}) \vee \cdots \vee C_t(\mathbf{v}) & \text{if } \beta = 1, \\ \neg\,[C_1(\mathbf{v}) \vee \cdots \vee C_t(\mathbf{v})] & \text{if } \beta = -1. \end{cases} \tag{C.13}$$

Thus, every image has a preimage, which further holds true for the equivalence relations imposed on the DNFs and DFCNs, proving that $\mathcal{G}$ is surjective.

Bijectivity of $\mathcal{G}$ follows from injectivity and surjectivity. See Appendix G of [1] for further details. □

C.5 Approximating $P(f)$ by sampling

We approximate $P(f)$ by Monte Carlo sampling from the uniform prior on network parameters, defined in Eq. (3.6):

$$W_{ij}^{(1)} \sim \mathrm{U}\{-1, 0, 1\}, \quad \beta \sim \mathrm{U}[-1, 1], \quad W^{(2)} = \beta \text{ (unless the corresponding clause is zero)}, \tag{C.14}$$

and estimate

$$P(f) = \Pr\left(f_\theta = f\right) = \frac{|\{\theta : f_\theta = f\}|}{2 \cdot 3^{n\,2^{n-1}}}. \tag{C.15}$$

Because $P(f)$ counts how many choices of θ implement f, it is exactly equivalent to the volume of parameter-space occupied by f. One may also view $P(f)$ as a Bayesian prior probability assigned to f.

In our Monte Carlo sampling to approximate $P(f)$, we draw 10^8 independent parameter samples. Any function f with $P(f) \lesssim 10^{-8}$ is vanishingly unlikely to appear in our search. For example, when $n = 4$, one computes

$$P\left(\text{parity}\right) = (2^3)!\, 3^{-4\cdot 2^3} \;\approx\; 2 \times 10^{-11}, \tag{C.16}$$

which explains why parity is never discovered (it lies three orders of magnitude below our sampling threshold). In fact, out of the $2^{2^4} = 65,536$ possible Boolean functions on 4 bits, we fail to encounter 631 of them even after 10^8 draws.

Figure C.2 plots the estimated $P(f)$ for $n = 3, 4, 5, 7$.

- **Top three rows:** $P(f)$ vs. $K_{DNF}(f)$, $K_\theta(f)$ and $K_C(f)$. The horizontal line at $P = 10^{-8}$ marks our effective sampling cutoff; finite-size artefacts appear for large $K_{DNF}(f)$, especially when $n = 7$. For $n = 3, 4$ there is a clear log-linear relationship between $P(f)$ and all complexity measures, with the difference between the maximum and minimum $P(f)$ at a fixed complexity small relative to the overall range of $P(f)$. For $n = 5, 7$, the total range of $P(f)$ is many orders of magnitude more than we can sample, but the upper bound $P(f) \sim 2^{-K_{DNF}(f)+O(1)}$ as observed in [64] still describes the distribution well.

- **Penultimate row:** $P(f)$ vs. $K_{LZ}(f)$, the Lempel–Ziv string complexity, as studied in [5, 1, 64]. The relation between $P(f)$ and $K_{LZ}(f)$ for $n = 3, 4$ is much weaker than for DNF complexity $K_{DNF}(f)$. This is what we might expect, given that DNF complexity is intuitively more appropriate in this case (as $K_{DNF}(f)$ is intricately connected to the architecture in a way $K_{LZ}(f)$ is not, see Appendix C.3). For $n = 5, 7$, we are unable to sample enough times to properly compare the distributions.

- **Bottom row:** $P(f)$ vs. rank $R(f)$ (where the most probable function has $R = 1$). The dashed orange line shows Zipf's law, $P(f) = (2^n \ln 2)^{-1} R(f)^{-1}$, which [243] identifies as the optimal prior for Bayesian learning.

Fig. C.3 shows the effect of the width of the hidden layer on the prior. We show widths $\alpha_w 2^{n-1}$ for $\alpha_w = 0.5, 1, 2, 4$, with the top two rows showing $n = 4$ and the bottom two rows showing $n = 5$. As the width increases, the probability that any input is True increases. We can use Lemma 12 to show that the probability that any input is False scales as $(1 - (2^n - 1)/3^n)^{\alpha_w 2^{n-1}}$. This expression decreases asymptotically to 0 as α_w increases. The **bottom row** in Fig. C.3 shows that the prior is not well-described by Zipf's law for $\alpha_w > 1$, indicating $\alpha_w = 1$ gives the optimal width for learning [243]. This is an interesting coincidence which we explore in the remainder of this section.

C.5.1 Finding the optimum width

As stated in the main text, we require a width of $\alpha_w 2^{n-1}$ with $\alpha_w \geq 1$ to guarantee full expressivity. However, Table 3.2 tells us that with this scaling, eventually the function space will be entirely dominated by the constant functions unless $\alpha_w \sim (3/4)^n$. Furthermore, it would be completely impractical to use a DFCN with width 2^{n-1} – by $n = 50$ to be fully expressive, you would need more than 10^{14} neurons.

So, how could we determine the optimum scaling? If Zipf's law is satisfied, the most frequent functions (in our work, the constant functions) should have $P(\text{const}) \sim 2^{-n}$. There are 2^n unique functions with $t = 1$ (a single True input), so these functions occupy a total space of size $\frac{1}{2^n \log 2} \sum_{i=3}^{i=2^n+2} i^{-1} \sim 2^{-n} n$, meaning the average 1-entropy function has probability $P(f_1^{(e)}) \sim n 2^{-2n}$. (Note that $i = 1, 2$ correspond to the two constant functions, and we ignore the flipped entropy functions that have only a single zero, which would be of the same order but only include an extra factor of 1/2, not affecting the overall approximation).

We assume that the presence of Zipf's law indicates an optimal prior. We argue this case in Appendix D.4, and will also make use of the derived results $P(\text{const}) \sim$

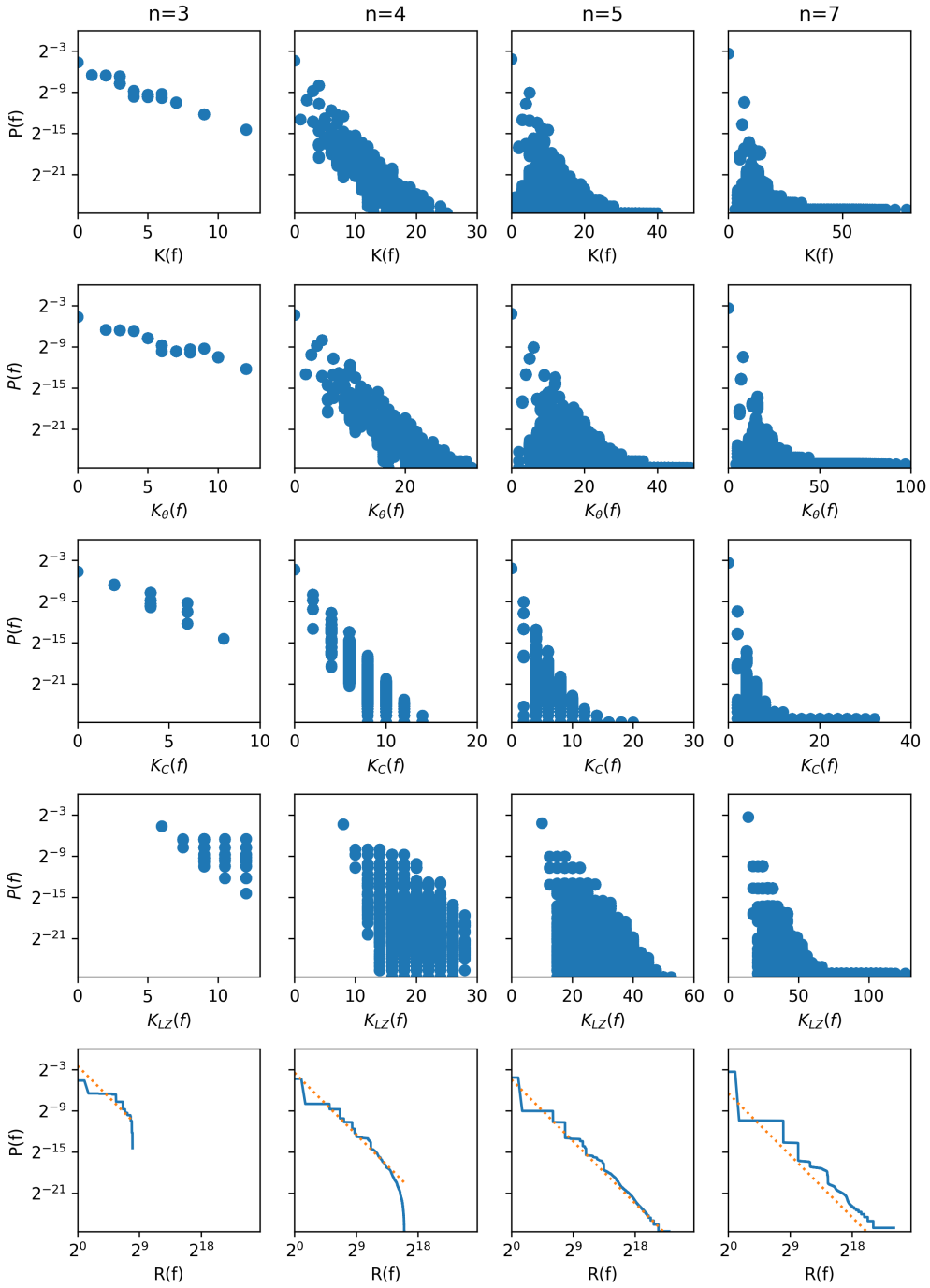


Figure C.2: Approximation of the prior probability $P(f)$ by sampling 10^8 functions from each prior for $n = 3, 4, 5$, and 7. **Top row:** $P(f)$ versus DNF complexity $K_{DNF}(f)$. Finite-size effects at $P(f) = 10^{-8}$ (sampling limit) produce artefacts at higher $K_{DNF}(f)$ in the $n = 7$ panel. **Second row:** $P(f)$ versus neural network norm $K_\theta(f)$. **Third row:** $P(f)$ versus clause complexity $K_C(f)$. **Fourth row:** $P(f)$ versus Lempel–Ziv complexity $K_{LZ}(f)$, as used in [5, 1, 64]. **Final row:** $P(f)$ versus rank $R(f)$, with dotted orange lines showing Zipf's law $P(f) = (2^n \ln 2)^{-1} R^{-1}$ [243].

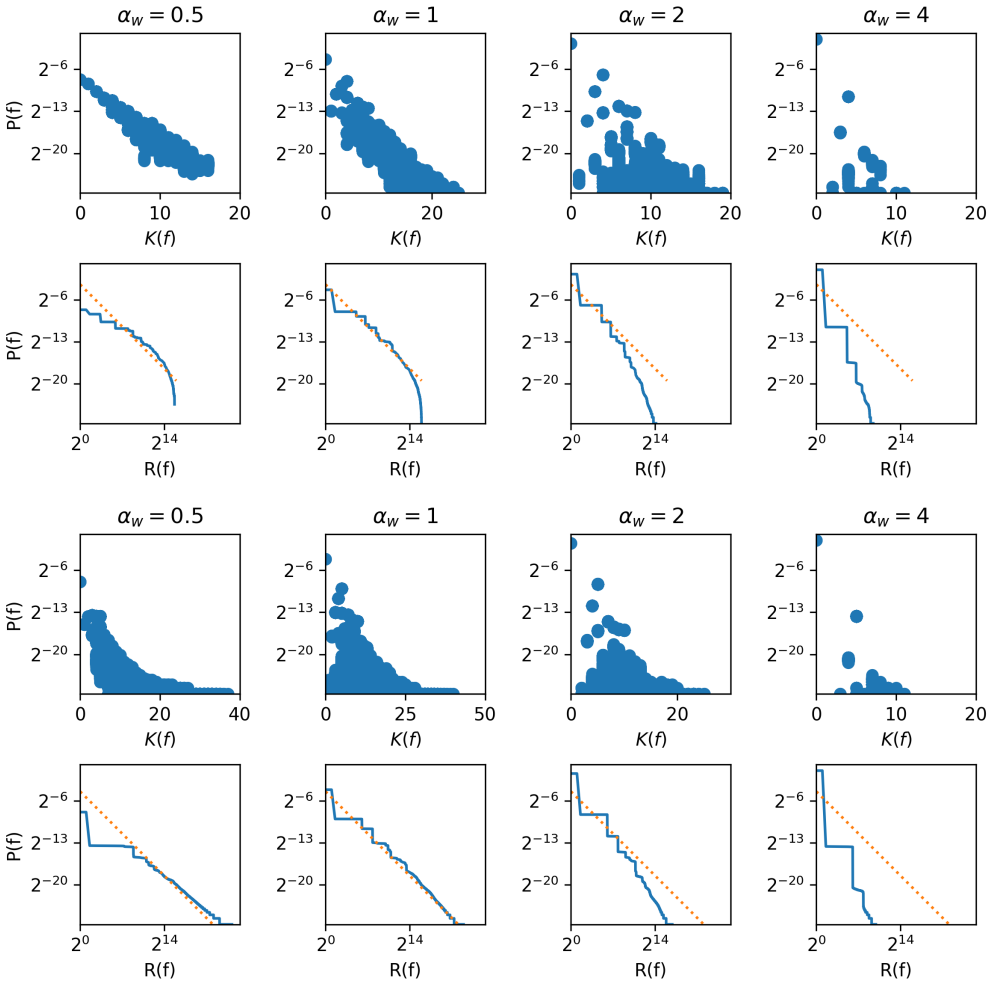


Figure C.3: Coverage of the prior probability $P(f)$ for **top two rows** $n = 4$ and **bottom two rows** $n = 5$ across three network widths $w \in \{1, 2, 4\} \times 2^{n-1}$, estimated by sampling 10^8 functions per prior. The $P(f)$ versus DNF complexity $K_{DNF}(f)$, illu plots illustrate how the constant function's probability mass grows with width. The plots showing $P(f)$ versus rank $R(f)$ (most probable is $R = 1$), with the dotted line marking Zipf's law $P(f) \propto R^{-1}$; larger widths exhibit marked departures from this scaling.

2^{-n} and $P(f_1^{(e)}) \sim n2^{-2n}$. We use the result from Eq. (C.65) with $p = \frac{2^n-1}{3^n}$,

$$P(f_1^{(e)}) \lesssim (1-p)^{\alpha_w 2^{n-1}} \tag{C.17}$$

$$\approx \exp\left(-\alpha_w 2^{n-1} p\right) \quad (p \ll 1) \tag{C.18}$$

$$\implies \quad \exp\left(-\alpha_w 2^{n-1} p\right) = n\, 2^{-n} \tag{C.19}$$

$$-\alpha_w 2^{n-1} p = \ln\left(n\, 2^{-n}\right) = -\, n \ln 2 + \ln n \tag{C.20}$$

$$\alpha_w 2^{n-1} = \frac{n \ln 2 - \ln n}{p} \sim n \ln 2 \, \left(\tfrac{3}{2}\right)^n \tag{C.21}$$

$$\alpha_w \sim 2n \ln 2 \left(\tfrac{3}{4}\right)^n. \tag{C.22}$$

We can also determine the width scaling to make the lower bound on $P(f^{(c)})$ scale as 2^{-n} by using the result in Lemma 12 that the probability that an input $\mathbf{v}$ is covered with probability $(1-p)^{\alpha_w 2^{n-1}}$, where $p = \frac{2^n-1}{3^n}$. Assuming independence and taking the Poisson approximation, we get

$$P(f^{(c)}) \approx \exp\left(-2^n e^{-\alpha_w 2^{n-1} p}\right) = 2^{-n} = \exp\left(-n \ln 2\right) \tag{C.23}$$

$$\implies \quad 2^n e^{-\alpha_w 2^{n-1} p} = n \ln 2 \tag{C.24}$$

$$e^{-\alpha_w 2^{n-1} p} = \frac{n \ln 2}{2^n} \tag{C.25}$$

$$-\alpha_w 2^{n-1} p = \ln\left(n \ln 2\right) - n \ln 2 \tag{C.26}$$

$$\alpha_w 2^{n-1} = \frac{n \ln 2 - \ln\left(n \ln 2\right)}{p} \approx \left(n \ln 2\right) \left(\tfrac{3}{2}\right)^n \tag{C.27}$$

$$\alpha_w \sim 2n \ln 2 \left(\tfrac{3}{4}\right)^n. \tag{C.28}$$

Both methods arrive at the same scaling for $\alpha_w \sim n \left(\frac{3}{4}\right)^n$. At small n, this scaling is very close to 1 – explaining why Fig. C.3 shows Zipf's law for $\alpha_w = 1$. To test the proposed optimal scaling more thoroughly, one would have to either make the scaling arguments above more precise or gather empirical evidence at larger n.

C.6 Results relating $P(f)$ to $K(f)$

In this appendix, we relate $P(f)$ to $K(f)$ for the various classes of functions discussed in the main text (Section 3.2.3): Constant, k-parity and t-entropy functions. We also briefly look at k-sparse functions, which are not studied in the main text. The width of the DFCN is $\alpha_w 2^{n-1}$, where $\alpha_w \geq 1$ for the network to be fully expressive.

C.6.1 Utility lemmas

Lemma 9 (Lower bound on $P(f)$)**.** *We can lower bound $P(f)$ using*

$$P(f) \geq \frac{1}{2} P(f \mid \beta) \tag{C.29}$$

as $P(\beta = 1) = P(\beta = -1) = \frac{1}{2}$

Lemma 10 (Lower bound on $P(f \mid \beta)$ using the minimum representation)**.** *Given a value for $\beta \in \{-1, 1\}$, we have the following lower bound on the conditional probability of f:*

$$P(f \mid \beta) \geq 3^{-nj} p! \left(\frac{p}{3^k}\right)^{\alpha_w 2^{n-1} - j}, \tag{C.30}$$

where j is the number of clauses, each of at most length k.

Proof. After permuting rows, the minimum representation of a function f can be encoded as follows,

$$W^{(1)} = \underbrace{\left[\begin{array}{c|c} \underbrace{A}_{j \times k} & B \\ \hline C & D \end{array}\right]}_{n} \Bigg\} \alpha_w 2^{n-1}, \tag{C.31}$$

$$W^{(2)} = \left[\underbrace{1, \ldots, 1}_{j}, \underbrace{0, \ldots, 0}_{\alpha_w 2^{n-1} - j}\right]^T \tag{C.32}$$

$$\beta \in \{-1, 1\} \tag{C.33}$$

where A contains p clauses each of at most length k, $B = 0$, $C = 0$ and $D = 0$. We can vary the clauses (rows) in A only up to permutation. How much freedom do we have to vary B, C, D? We must set $B = 0$, otherwise k would not be the maximum length of the minimal representation. If every clause in C is a copy of one in A and $B = 0$, we can let D be anything without affecting the overall function. This gives us the following lower bound:

$$P(f \mid \beta) \geq \underbrace{j! 3^{-jk}}_{A} \times \underbrace{3^{-j(n-k)}}_{B} \times \underbrace{\left(j/3^k\right)^{\alpha_w 2^{n-1} - j}}_{C} \times \underbrace{1}_{D} \tag{C.34}$$

$$\geq 3^{-nj} j! \left(\frac{j}{3^k}\right)^{\alpha_w 2^{n-1} - j}. \tag{C.35}$$

□

Lemma 11. *Denote $\mathcal{N}$ the set of all possible clauses given a $\beta \in \{-1, 1\}$, with $N = |\mathcal{N}| = 3^n$. Let $M = \alpha_w 2^{n-1}$ be the number of clauses drawn i.i.d. uniformly from $\mathcal{N}$. Consider a subset $Q \subseteq \mathcal{N}$ of size $q = |Q|$ containing clauses that we must*

have at least one copy of, and a disjoint subset of clauses $R \subseteq \mathcal{N}$ of size $r = |R|$ that we must not have. We can then lower bound $P(f)$ as follows:

$$P(f \mid \beta) = P = \Pr\Big(\textit{all } x \in Q \textit{ appear at least once, and no } y \in R \textit{ appears}\Big), \tag{C.36}$$

where

$$P = \sum_{i=0}^{q} (-1)^i \binom{q}{i} \left(1 - \tfrac{r+i}{N}\right)^{\alpha_w 2^{n-1}}, \tag{C.37}$$

and the union-bound lower bound is given by

$$P(f \mid \beta) \geq \left(\tfrac{N-r}{N}\right)^{\alpha_w 2^{n-1}} \left[1 - q\Big(1 - \tfrac{1}{N-r}\Big)^{\alpha_w 2^{n-1}}\right]. \tag{C.38}$$

In the rest of this section, we will rely on the first term in Eq. (C.38), but as the second term is often very loose, we will often bound it below using an alternative task-specific bound.

Proof. Let

$$A = \{\text{every } x \in Q \text{ appears}\}, \quad B = \{\text{no draw lies in } R\}. \tag{C.39}$$

Then

$$P = \Pr(A \cap B) = \Pr(A \mid B) \Pr(B). \tag{C.40}$$

Since each of the M draws must avoid R,

$$\Pr(B) = \left(\tfrac{N-r}{N}\right)^{M}. \tag{C.41}$$

Conditioned on B, draws are uniform on the remaining $N - r$ symbols, and by the inclusion–exclusion principle

$$\Pr(A \mid B) = \sum_{i=0}^{q} (-1)^i \binom{q}{i} \left(\tfrac{(N-r)-i}{N-r}\right)^{M}. \tag{C.42}$$

Combining these and noting $\left(\frac{N-r}{N}\right)^{M} \left(\frac{N-r-i}{N-r}\right)^{M} = (\frac{N-r-i}{N})^{M}$ yields the exact sum. Truncating after $i = 1$ gives us the union-bound of

$$\Pr(A \mid B) \geq 1 - q \left(1 - \frac{1}{N-r}\right)^{M}, \tag{C.43}$$

from which we obtain the union-bound on P by multiplying with $\Pr(B)$.

□

Lemma 12 (Probability of input $\mathbf{v}$ being True)**.** *Given a fixed input* $\mathbf{v} \in \{0,1\}^n$*, the probability that a randomly sampled clause* C *covers* $\mathbf{v}$ *(i.e., is True on* $\mathbf{v}$*) is*

$$P(C(\mathbf{v}) = 1) = \frac{2^n - 1}{3^n}. \tag{C.44}$$

Given a DFCN of width $\alpha_w 2^{n-1}$*, the probability that any particular input* $\mathbf{v}$ *is True is*

$$P(f(\mathbf{v}) = 1 \mid \beta = 1) = 1 - \left(1 - \frac{2^n - 1}{3^n}\right)^{\alpha_w 2^{n-1}} \tag{C.45}$$

Moreover, the leading-order term for large n *is*

$$P(f(\mathbf{v}) = 1 \mid \beta = 1) \sim \frac{\alpha_w}{2}\left(\tfrac{4}{3}\right)^n. \tag{C.46}$$

Proof. A clause is True on input x if for each variable in the input, the corresponding entry of a clause in DFCN representation is 1 if $x_i = 1$, is -1 if $x_i = 0$ or is 0 (but ignoring the all 0s case, which is considered False). This means we have $2^n - 1$ clauses that satisfy this criterion. Divide by the total number of clauses, 3^n for Eq. (C.44). Since all clauses are drawn independently, the probability that all clauses give False on a random input $\mathbf{v}$ is

$$P(f(\mathbf{v}) = 0 \mid \beta = 1) = \left(1 - \frac{2^n - 1}{3^n}\right)^{\alpha_w 2^{n-1}}, \tag{C.47}$$

from which Eq. (C.45) follows. □

C.6.2 Function class: constant

Denote the class of constant functions as $f^{(c)}$, which includes all functions where the output either always gives True or always gives False, regardless of the input.

Lemma 13 (Lower bound for $P(f^{(c)})$)**.** *We can bound either constant function,* $P(f^{(c)})$ *with*

$$P(f^{(c)}) \geq \frac{1}{2}\sum_{k=0}^{2^n}(-1)^k\binom{2^n}{k}\left(1 - kp\right)^{\alpha_w 2^{n-1}}, \tag{C.48}$$

where $p = \left(\frac{2^n-1}{3^n}\right)$*. When truncating after* $k = 1$*, we obtain the lower bound*

$$P(f^{(c)}) \geq \tfrac{1}{2}\left(1 - 2^n(1-p)^{\alpha_w 2^{n-1}}\right) \tag{C.49}$$

Substituting $p = \frac{2^n-1}{3^n} \sim \left(\frac{2}{3}\right)^n$, *we get*

$$P(f^{(c)}) \gtrsim \tfrac{1}{2}\left(1 - \exp\Big(n\ln 2 - \tfrac{\alpha_w}{2}(4/3)^n\Big)\right) \quad (n \to \infty). \tag{C.50}$$

Proof. Let $n \in \mathbb{N}$, $M = \alpha_w 2^{n-1}$, and $p = \frac{2^n-1}{3^n}$ (Lemma 12). We fix $\beta = 1$ and aim to bound the function that returns True for all inputs. We label the 2^n Boolean inputs by $\mathbf{v} \in \{0,1\}^n$, and for each $\mathbf{v}$ let

$$A_{\mathbf{v}} = \Big\{\text{no clause covers } \mathbf{v}\Big\}, \qquad \Pr(A_{\mathbf{v}}) = (1-p)^M. \tag{C.51}$$

Then by the principle of inclusion-exclusion, the probability that every input is covered by at least one clause (i.e. no $A_{\mathbf{v}}$ occurs) is exactly

$$P(f^{(c)} \mid \beta = 1) = \Pr\Big(\bigcap_{\mathbf{v}\in S} A_{\mathbf{v}}^c\Big) = 1 - \Pr\Big(\bigcup_{\mathbf{v}\in S} A_{\mathbf{v}}\Big) = \sum_{k=0}^{2^n}(-1)^k \sum_{\substack{S\subseteq\{0,1\}^n \\ |S|=k}} \Pr\Big(\bigcap_{\mathbf{v}\in S} A_{\mathbf{v}}\Big). \tag{C.52}$$

Moreover, for any fixed S of size k, one has

$$\Pr\Big(\bigcap_{\mathbf{v}\in S} A_{\mathbf{v}}\Big) = \Big(1 - p_S\Big)^M, \tag{C.53}$$

where

$$p_S = \Pr\Big(\text{a single random clause covers at least one } \mathbf{v} \in S\Big)$$

Furthermore, by the union bound on the single-clause covering probabilities,

$$p_S \le \sum_{x\in S} p = kp, \tag{C.54}$$

so that

$$\Pr\Big(\bigcap_{\mathbf{v}\in S} A_{\mathbf{v}}\Big) = \Big(1 - p_S\Big)^M \ge (1-kp)^M. \tag{C.55}$$

Substituting into the inclusion–exclusion sum gives the valid lower bound

$$P(f^{(c)} \mid \beta = 1) \ge \sum_{k=0}^{2^n}(-1)^k \binom{2^n}{k}\Big(1 - kp\Big)^M. \tag{C.56}$$

Truncating after $k = 1$ gives $P(f^{(c)} \mid \beta = 1) \ge 1 - 2^n(1-p)^M$, and approximating p to be small, so that $(1-p)^M \approx e^{-Mp}$, yields the final estimate for P,

$$P(f^{(c)} \mid \beta = 1) \ge 1 - 2^n(1-p)^{\alpha_w 2^{n-1}} \approx 1 - 2^n e^{-\alpha_w 2^{n-1} p}. \tag{C.57}$$

Using Lemma 9, we thus have $P(f^{(c)}) \ge \frac{1}{2}P(f^{(c)} \mid \beta = 1)$.

□

C.6.3 Function class: entropy

Consider the class of functions with t 1s and $2^n - t$ 0s. We call this function t-entropy and denote it with $f_t^{(e)}$. If t is small, the function is simple (consistent with the intuition that low-entropy functions are simple). However, the converse does not hold: some high-entropy functions require a very small number of clauses (e.g. 1-parity needs just one clause: x_1).

Lemma 14. *Given a boolean function on n variables with t 1s and $2^n - t$ 0s, we denote R the set of forbidden clauses with $r = |R|$ and $N = 3^n$ the total number of possible clauses (Lemma 11). The fraction of clauses that would flip the function value if appended to its DNF is bounded below by:*

$$\frac{r}{N} \geq (2/3)^{n-\lfloor \log_2(2^n - t) \rfloor} \tag{C.58}$$

Proof. Let $f : \{0,1\}^n \to \{0,1\}$ have exactly $\tilde{t} = 2^n - t$ inputs $\mathbf{v}^{(i)} \in \{\mathbf{v}^{(1)}, ..., \mathbf{v}^{(\tilde{t})}\}$ for which $f(\mathbf{v}^{(i)}) = 0$, with $1 \leq \tilde{t} \leq 2^n - 1$. By filling the largest possible d-dimensional subspace of the n-dimensional hypercube, given by $d \leq \lfloor \log_2 \tilde{t} \rfloor$, this corresponds to finding the maximal correlation between these inputs, which minimises the set of not allowed clauses R.

Given a maximally filled d-dimensional subspace, the remaining $n - d$ bits for these points must be the same. Since the subspace is fully filled, all possible combinations of inputs in this subspace are exhausted, meaning that unless all entries in the DFCN representation of a clause are zero (which always gives False), one of these subspace inputs must give True. Thus, we need the probability that all remaining $n - d$ entries in a DFCN clause either match the corresponding remaining input bit (1 if $x_i = 1$, -1 if $x_i = 0$) or are 0, giving

$$P = \left(\frac{2}{3}\right)^{n-d} - 1. \tag{C.59}$$

(The -1 comes from the all zeros clause.) Substituting the bound on d and taking n to be large gives us

$$P = \frac{r}{N} \geq \left(\frac{2}{3}\right)^{n-\lfloor \log_2 \tilde{t} \rfloor}. \tag{C.60}$$

□

Lemma 15 (Upper bound for t-entropy)**.** *We can upper bound $P(f_t^{(e)})$ with the following*

$$P(f_t^{(e)} \mid \beta) \lesssim \begin{cases} \exp\left(-\alpha_w 2^{n-1}(2/3)^{n-\lfloor \log_2(2^n - t)\rfloor}\right) & \text{if } \beta = 1, \\ \exp\left(-\alpha_w 2^{n-1}(2/3)^{n-\lfloor \log_2 t \rfloor}\right) & \text{if } \beta = -1. \end{cases} \tag{C.61}$$

Proof. Let $f_t^{(e)} : \{0,1\}^n \to \{0,1\}$ have exactly t 1s, with $1 \leq t \leq 2^n - 1$. For the case of $t \leq 2^{n-1}$, we take $\beta = 1$ (Appendix C.4), giving us $\tilde{t} = 2^n - t$ inputs $\mathbf{v}^{(i)} \in \{\mathbf{v}^{(1)}, ..., \mathbf{v}^{(\tilde{t})}\}$ for which $f_t^{(e)}(\mathbf{v}^{(i)}) = 0$. (For $t > 2^{n-1}$ we take $\beta = -1$ and simply replace $\tilde{t}$ with t.) Given the set R, with $r = |R|$, of all forbidden clauses which would flip a 0 output to a 1, and $N = 3^n$ total clause options, a valid network must sample all M clauses outside of R, giving us an upper bound,

$$P\Big(f_t^{(e)} \mid \beta = 1\Big) \leq \left(1 - \tfrac{r}{N}\right)^M \lesssim \exp\Big(-Mr/N\Big). \tag{C.62}$$

Using the result in Lemma 14, we have $r/N \geq (2/3)^{n-\lfloor \log_2 \tilde{t} \rfloor}$. Substituting $M = \alpha_w 2^{n-1}$ into (C.62) then gives

$$P\Big(f_t^{(e)} \mid \beta = 1\Big) \leq \left(1 - (2/3)^{n-\lfloor \log_2(2^n - t)\rfloor}\right)^{\alpha_w 2^{n-1}} \approx \exp\left(-\alpha_w 2^{n-1}(2/3)^{n-\lfloor \log_2(2^n - t)\rfloor}\right). \tag{C.63}$$

□

Lemma 16 (Bounds on t-entropy with $t = 1$)**.** *For a function with a single True output* $\mathbf{v}$ *and all else False,*

$$P(f_1^{(e)} \mid \beta = -1) \leq \left(1 - \frac{2^n - 1}{3^n}\right)^{\alpha_w 2^{n-1}} \tag{C.64}$$

We also have a lower bound

$$P(f_1^{(e)} \mid \beta = -1) \geq 3^{-n^2} \left(1 - \frac{2^n - 1}{3^n}\right)^{\alpha_w 2^{n-1} - n}. \tag{C.65}$$

The leading order behaviour of this (for constant α_w) is

$$P(f_1^{(e)} \mid \beta = -1) \gtrsim \exp\left(-\frac{\alpha_w}{2}\left(\frac{4}{3}\right)^n\right) \tag{C.66}$$

Proof. We set $\beta = -1$, so that we are now solving for a function where only one input $\mathbf{v^{(1)}} \in \{0,1\}^n$ has an output of 0. To prove the upper bound, we simply require no clause to be True on input $\mathbf{v^{(1)}}$.

To prove the lower bound, we can use the fact that the minimal representation of this function is given by a weight $W^{(1)}$ with only non-zero entries in the main diagonal,

$$W^{(1)}_{ii} = \begin{cases} +1 & \text{if } v^{(1)}_i = 0, \\ -1 & \text{if } v^{(1)}_i = 1. \end{cases} \tag{C.67}$$

Note this is the opposite of the typical construction where we assign $+1$ if the input bit is 1 and -1 if the input bit is 0. Provided none of the $2^n - 1$ clauses that make $f^{(e)}_1(\mathbf{v})$ output True are drawn (Lemma 12), we can lower bound $P(f^{(e)}_1)$ by setting the first n rows of $W^{(1)}$ as described above (which would happen with probability 3^{-n^2}) and require the rest of the rows to exclude any of the $2^n - 1$ forbidden clauses. □

$P(f)$ for a random function with fixed t

In Appendix C.6.3, we upper bounded $P(f^{(e)}_t)$ by computing the minimum number of forbidden clauses at every entropy class. We know there can be a huge range in $P(f^{(e)}_t)$, the best example being $t = 2^{n-1}$. Both 1-parity and n-parity have 2^{n-1} 1s, yet $P(f^{(p)}_n) \sim (3/2)^{-\alpha_w n 2^{n-1}}$ and $P(f^{(p)}_1) \sim (3/2)^{-\alpha_w 2^{n-1}}$. The bounds must satisfy $P(f^{(p)}_1)$ and are therefore far too loose for $f^{(p)}_n$.

We did not, however, determine how $P(f^{(e)}_t)$ should scale for the typical function with t 1s (i.e. a uniformly sampled function from the set of functions with exactly t 1s). Consider a function f, and define the number of False outputs $\tilde{t} = 2^n - t$. The probability of drawing a clause C which is True on an input $\mathbf{v}$ for which $f(\mathbf{v})$ outputs False is $p = (2^n - 1)/3^n$ (Lemma 12). Then with R as the set of all forbidden clauses and $r = |R|$, the probability of drawing a forbidden clauses $C \in R$ is

$$\Pr(C \in R) = \frac{r(n,t)}{3^n} = 1 - (1-p)^{2^n - t}, \tag{C.68}$$

assuming independence. Fig. C.4 shows that this assumption is only valid for $\tilde{t} = O(n)$. As $\tilde{t}$ increases, Eq. (C.68) overestimates $P(C \in R)$. The empirical

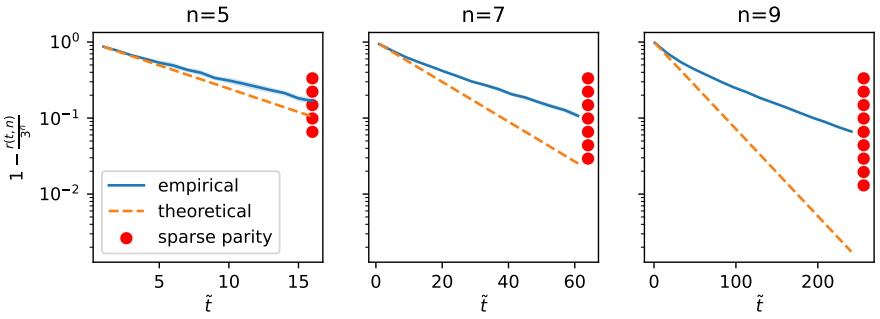


Figure C.4: Fraction of accepted clauses $1 - \frac{r(n,t)}{3^n}$ versus the number of zeros, $\tilde{t}$, in an n-variable Boolean function. Error bars show 1 standard deviation. The theoretical line uses Eq. (C.68), and assumes independence. This is only a good assumption for low $\tilde{t}$. As expected, when $\tilde{t}$ is small (the function is almost entirely True), almost all clauses are accepted without changing the function. The red dots indicate all k-parity functions for $1 \leq k \leq n$.

data was generated by uniformly sampling 20 functions at fixed t and calculating r by exhaustive enumeration for those functions. This gives us an idea of $r(n,t)$ for the "typical" function with t 1s. We also plotted the k-parity functions for $1 \leq k \leq n$ (which all have $t = 2^{n-1}$).

C.6.4 Function class: parity

Let $f_k^{(p)} : \{0,1\}^n \to \{0,1\}$ be the k-parity function on the first k bits:

$$f_k^{(p)}(\mathbf{v}) = \sum_{i=1}^{k} v_i \pmod 2, \qquad 1 \leq k \leq n. \tag{C.69}$$

As one checks directly, any representation of $f_k^{(p)}$ in our random network model must use exactly $j = 2^{k-1}$, distinct clauses of length k, and no shorter set of clauses can realise parity. With $M = \alpha_w 2^{n-1}$ and $N = 3^n$ we can construct lower and upper bounds with the following two propositions.

Proposition C.6.1 (Lower bound for k-parity)**.** *Using Lemma 10, with the minimum representation size $j \times k$, set $j = 2^{k-1}$. Then,*

$$P(f_k^{(p)}) \geq 3^{-n2^{k-1}} (2^{k-1})! \left(\frac{2^{k-1}}{3^k}\right)^{\alpha_w 2^{n-1} - 2^{k-1}}. \tag{C.70}$$

Applying Stirling's inequality $j! \geq \sqrt{2\pi j}(j/e)^j$*, in the large* n *limit we have*

$$P(f_k^{(p)}) \geq 3^{-n2^{k-1}} \sqrt{2\pi 2^{k-1}} \left(\frac{2^{k-1}}{e}\right)^{2^{k-1}} \left(\frac{2^{k-1}}{3^k}\right)^{\alpha_w 2^{n-1} - 2^{k-1}} \tag{C.71}$$

$$\geq \exp\Big\{\Big(-\alpha_w 2^{n-1}(k\ln(3/2) + \ln 2) + O(n2^{k-1})\Big)\Big\}. \tag{C.72}$$

Proposition C.6.2 (Upper bound for k-parity)**.** *Exactly* 2^{k-1} *of the* 3^k *clauses of length* $\leq k$ *implement* k*-parity, so at each of the* $\alpha_w 2^{n-1}$ *draws the chance of choosing an admissible clause is at most* $\frac{2^{k-1}}{3^k}$*. Therefore*

$$P(f_k^{(p)}) \leq \left(\tfrac{2^{k-1}}{3^k}\right)^{\alpha_w 2^{n-1}}. \tag{C.73}$$

At large n*,*

$$P(f_k^{(p)}) \leq \exp\Big\{\Big(-\alpha_w 2^{n-1}(k\ln(3/2) + \ln 2)\Big)\Big\} \tag{C.74}$$

The bounds above match up to constants, so

$$P\Big(f_k^{(p)}\Big) = e^{-\Theta(\alpha_w k 2^{n-1})}. \tag{C.75}$$

C.6.5 Function class: sparse

A Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is called k-sparse if it depends on exactly k of its n input bits (say $x_1, \ldots, x_k$) and is independent of the remaining $n-k$ bits. Equivalently, for every fixed $(x_1, \ldots, x_k)$, flipping any of the last $n-k$ coordinates does not change f. In the ordered listing of the truth table (Fig. 3.2), f then repeats its 2^k-bit pattern 2^{n-k} times.

It is hard to come up with bounds for k-sparse functions that are meaningful, as the complexity range for a given k can be very large. The most complex k-sparse in each class is k-parity. At the other end of the spectrum, there are functions where the minimum representation (Eq. (C.31)) has $A = I_k$ (the identity matrix with dimension k). We could lower bound this type of function by requiring I_k to exist (with probability 3^{-nk}), and that the first k elements of the rest of the clauses must not contain exclusively -1 and 0 (but we permit all 0s). The probability that this

happens is $\left(\frac{2}{3}\right)^k - 3^{-n}$. We can multiply this by the total number of clause options, $N = 3^n$, to give us $r = 3^n \left(\frac{2}{3}\right)^k - 1$ and then use Lemma 11 to get,

$$3^{-nk}\left(1 - \left(\frac{2}{3}\right)^k + 3^{-n}\right)^{\alpha_w 2^{n-1}-k} \leq P(f_k^{(s)}) \leq \left(1 - \left(\frac{2}{3}\right)^k + 3^{-n}\right)^{\alpha_w 2^{n-1}}, \quad \text{(C.76)}$$

which to leading order scales as $\exp\left\{\left(-\alpha_w 2^{n-1}\left(\frac{2}{3}\right)^k\right)\right\}$, decaying slower than k-parity for large k. (The upper bound comes from rejecting all forbidden clauses.)

C.7 Trained networks

C.7.1 Generating datasets

Each function is a Boolean map $f : \{0,1\}^n \to \{0,1\}$, stored as an n-dimensional binary input vector and a scalar output. To ensure reproducibility, we set a fixed random seed at the start of generation. Given a training set size m, we randomly shuffle all 2^n possible inputs and take the first m as training examples; the remaining $2^n - m$ points form the test set.

We train DFCNs on the following three functions

1. **k-parity:** Choose a random subset $S \subseteq \{1, \ldots, n\}$ of size k, and define: $f(x) = \bigoplus_{i \in S} x_i$.

2. **t-entropy**: Select t input points uniformly at random from the 2^n possibilities and assign $f(x) = 1$ on those points (all others map to 0), yielding functions of fixed Hamming weight t.

3. **k-sparse:** Generate a random binary string $s \in \{0,1\}^{2^k}$, then tile it 2^{n-k} times to form the full function string of length 2^n.

Note that we do not study k-sparse functions in the main text. These are functions generated by repeated patterns of length 2^k.

In our experiments, we fix $n = 7$. The parameter grids are:

- k-parity: $k \in \{1, 2, \ldots, 7\}$.

- t-entropy: $t \in \{0, 4, 8, 16, 32, 35, 64\}$.

- k-sparse tile lengths $l \in \{2, 4, 5, 8, 13, 16, 32\}$.

Note that the repeating patterns with $l = 5, 13$ do not generate k-sparse functions (unlike the other lengths l given above). We include them as they have low LZ complexity but not low DNF complexity (see Appendix C.3), and this example is useful in demonstrating why $K_{DNF}(f)$ is a better measure of complexity than $K_{LZ}(f)$.

C.7.2 Metropolis-Hastings algorithm

While we could in theory use an SGD-like algorithm (Algorithm 4), which aims to find the direction of steepest descent and move there, it is not Bayesian, and enumerating the entire neighbourhood rapidly increases exponentially in computational complexity as n increases. In this section, we define a Metropolis-Hastings algorithm that is Bayesian, and does not suffer from these scaling problems.

Let $\theta = (W^{(1)}, W^{(2)})$ denote the parameter vector of weights in a DFCN, with

$$W^{(1)} \in \{-1, 0, 1\}^{n \times \alpha_w 2^{n-1}}, \quad W^{(2)} \in \{0, 1\}^{\alpha_w 2^{n-1}}.$$

We write $f(\theta; S)$ for the network's predictions on a dataset S, and $L\Big(f(\theta; S)\Big)$ for its empirical loss (e.g. classification error) on S. We also use the ℓ_1-norm regulariser

$$\|\theta\|_1 \;=\; \|W^{(1)}\|_1 + \|W^{(2)}\|_1.$$

Target (posterior) density. Given inverse-temperature $\kappa > 0$ and weight-decay factor $\lambda \geq 0$, we seek to sample from

$$\pi(\theta) \;\propto\; \exp\Big[-\kappa L\Big(f(\theta; S)\Big) - \lambda \|\theta\|_1\Big].$$

Proposal distribution. For any current state θ, its 1-hop neighbourhood is

$$\mathcal{N}(\theta) \;=\; \{\theta' \colon d(\theta, \theta') = 1\},$$

where $d(\theta, \theta')$ is the Hamming distance between discrete parameters. We use the uniform proposal

$$g(\theta \to \theta') \;=\; \begin{cases} \dfrac{1}{|\mathcal{N}(\theta)|}, & \text{if } \theta' \in \mathcal{N}(\theta), \\ 0, & \text{otherwise.} \end{cases}$$

Algorithm 2 writes down the process explicitly.

This algorithm trains well with $\kappa = 1000$ and $\lambda = 0$ – increasing λ to 0.1 had limited effect except to destabilise early training. Fig. C.5 shows the outcomes of training a DFCN with Algorithm 2 on different function classes. For entropy and repeated functions, we see that adding weight decay greatly improves performance, especially when the size of the training set m is smaller. As discussed in Section 3.3.3, weight decay does not provide any significant advantages when trying to learn highly complex functions such as 7-parity.

Algorithm 2 Metropolis–Hastings optimisation for DFCNs

Initialise:
$W^{(1)} \sim \{-1, 0, 1\}^{n \times \alpha_w 2^{n-1}}$, $\beta \sim \{-1, 1\}$, $W^{(2)} \sim \{0, \beta\}^{\alpha_w 2^{n-1}}$
$b1 \leftarrow b1(W^{(1)})$, $b2 \leftarrow b2(\beta)$ {Biases are functions of weights, see Definition 6}
Input: Training set S, batch size $b = |S|$, iterations N {full batch if Bayesian}
Hyperparams: $\kappa > 0,\ \lambda \geq 0$
Initialise: $\theta^{(0)}$ uniformly in parameter space
for $t = 1, \ldots, N$ **do**
  Sample minibatch $S_t \subset S$, $|S_t| = b$
  Propose $\theta' \sim g(\theta^{(t-1)} \to \cdot)$
  Compute losses $L_{\text{old}} = L\Big(f(\theta^{(t-1)}; S_t)\Big),\ L_{\text{new}} = L\Big(f(\theta'; S_t)\Big)$
  Compute acceptance probability

$$\alpha \ = \ \min\Big\{1,\ \exp\Big[\kappa\,(L_{\text{old}} - L_{\text{new}}) \ + \ \lambda\,(\|\theta^{(t-1)}\|_1 - \|\theta'\|_1)\Big]\Big\}$$

  Draw $u \sim \text{Uniform}(0, 1)$
  if $u < \alpha$ **then**
    $\theta^{(t)} \leftarrow \theta'$
  else
    $\theta^{(t)} \leftarrow \theta^{(t-1)}$
  end if
end for

C.7.3 Min norm Oracle algorithm

We also define an Oracle algorithm, which computes the minimal complexity DNF compatible with the training set. This is obtained by exhaustive search and is only possible for small enough n. Since this always returns the minimum norm DFCN

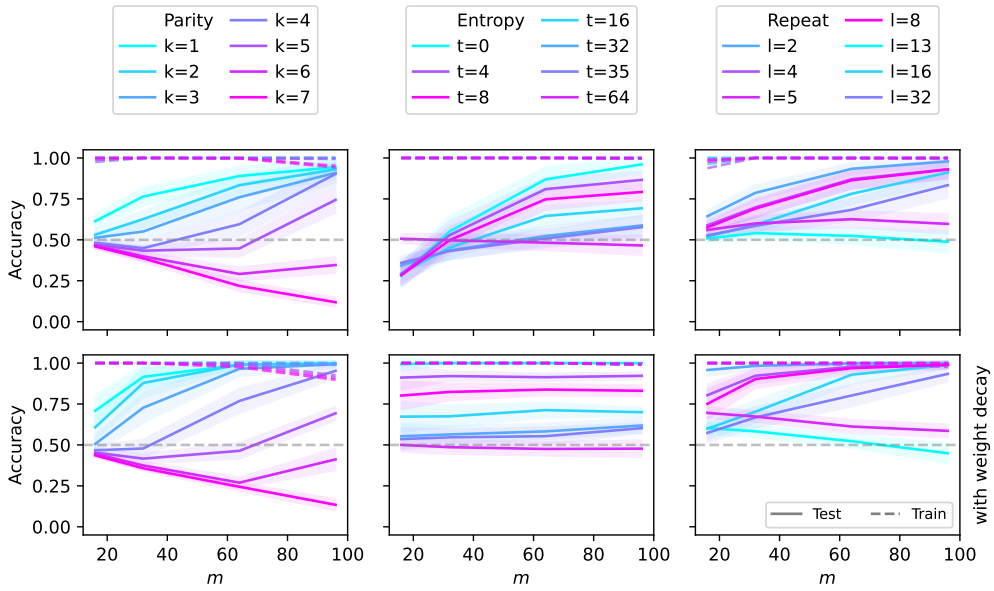


Figure C.5: MCMC algorithm (Algorithm 2 with $\kappa = 1000$) trained on different targets from the $n = 7$ dataset. Each column shows a different function class – parity, entropy and repeat. See Appendix C.7.1 for full experimental details and a description of each function type. As with the SGD-like algorithm shown in Fig. C.7, weight decay (bottom row, $\lambda = 0.01$) outperforms no weight decay (top row $\lambda = 0$), especially for small training set size m.

solution, this acts as a good baseline to compare other algorithms to, as we can see how close our trained function is to having the optimal minimal complexity.

Algorithm 3 Min norm Oracle

Initialise:
True inputs $P \leftarrow \{\}$ {Inputs for which $f(\mathbf{v}) = 1$}
DNF {Minimum DNF}
Input: Training data S, test data T
for $\mathbf{s} \in S$ **do**
  if $f(\mathbf{s}) = 1$ **then**
    $P \leftarrow P \cup \{\mathbf{s}\}$
  end if
end for
DNF $\leftarrow$ SOPform(variables=$[x_1, \ldots, x_n]$, minterms=P, dontcares=T)

The training algorithm is shown in Algorithm 3. The SOPform function comes from the sympy logic module [244] and finds the minimal DNF expression for a given set of inputs that output True. It takes the following arguments:

- **variables:** A list of symbols denoting the literals in the DNF.

- **minterms:** All inputs for which the output of the expression should give True.

- **dontcares:** All inputs for which we don't care about the output (i.e. the test data).

See Fig. C.6 for data.

C.7.4 SGD-like algorithm

Despite not being Bayesian, we also trained with an SGD-like algorithm, which worked well for small n in which the algorithm is tractable. In Algorithm 4, we begin by randomly initializing the first-layer weights $W^{(1)} \in \{-1, 0, 1\}^{n \times \alpha_w 2^{n-1}}$ and second-layer weights $W^{(2)} \in \{0, 1\}^{\alpha_w 2^{n-1}}$, and computing the dependent biases via $b1 = b1(W^{(1)})$ and $b2 = b2(\beta)$ (see Definition 6). Over N iterations, we draw a minibatch S_t of size b from the training set, evaluate the current network accuracy on S_t, and enumerate all one-hop neighbours of $(W^{(1)}, W^{(2)})$. For each neighbour, we recompute its biases and measure its batch accuracy, then collect the subset $\mathcal{N}_{\text{best}}$ of weights achieving the highest performance. With probability p we choose the neighbour from $\mathcal{N}_{\text{best}}$, which minimises the ℓ_1 norm $\|W^{(1)}\|_1 + \|W^{(2)}\|_1$ (thus encouraging sparsity), and otherwise select uniformly at random from $\mathcal{N}_{\text{best}}$. The chosen weights replace $(W^{(1)}, W^{(2)})$, their biases are updated, and the procedure repeats. Upon completion, the algorithm returns a two-layer DFCN that is locally optimised for the training data.

Fig. C.7 shows this algorithm trained on a DFCN with $n = 7$ over a wide range of functions. We see that the performance is very similar to Algorithm 2. However, since this SGD-like algorithm does not scale well, we would instead opt to use the MCMC algorithm for large n.

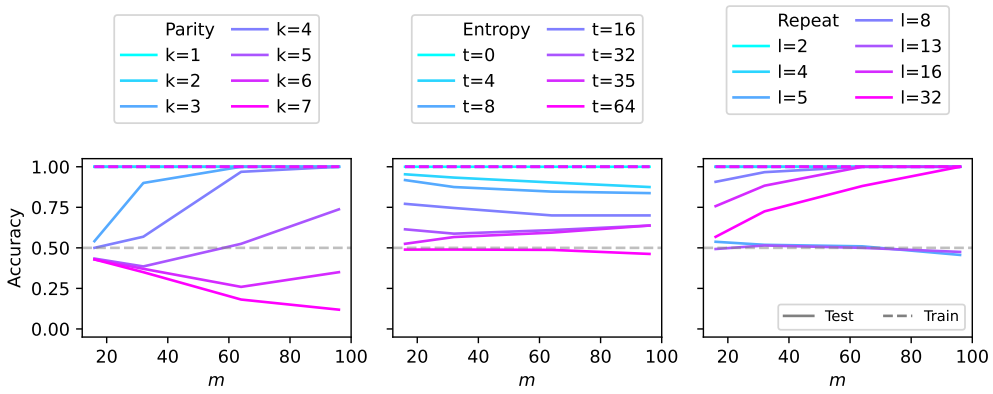


Figure C.6: The oracle trained on different targets from the $n = 7$ dataset. Each column shows a different function class – parity, entropy and repeat. See Appendix C.7.1 for full experimental details and a description of each function type.

Algorithm 4 SGD-like optimisation for DFCNs

Initialise:
$W^{(1)} \sim \{-1, 0, 1\}^{n \times \alpha_w 2^{n-1}}$, $\beta \sim \{-1, 1\}$, $W^{(2)} \sim \{0, \beta\}^{\alpha_w 2^{n-1}}$
$b1 \leftarrow b1(W^{(1)})$, $b2 \leftarrow b2(\beta)$ {Biases are functions of weights, see Definition 6}
Input: Training data S, test data T, batch size b, probability p, number of iterations N
for $t = 1$ to N **do**
  Sample a batch $S_t \subset S$ of size b
  Compute current accuracy a_{curr} on S_t
  Generate 1-hop neighbours of $W^{(1)}$ and $W^{(2)}$
  for each neighbour $(W^{(1)'}, W^{(2)'})$ **do**
    for $\beta \in \{-1, 1\}$ **do**
      $b1' \leftarrow b1(W^{(1)'})$, $b2' \leftarrow b2(\beta)$ {We leave β fixed at 1 in our experiments}
      Compute accuracy a' of $(W^{(1)'}, b1', W^{(2)'}, b2')$ on S_t
    end for
  end for
  Identify neighbour set $\mathcal{N}_{\text{best}}$ with best accuracy
  if $r \sim U[0, 1] < p$ **then**
    Select $(W^{(1)\star}, W^{(2)\star})$ from $\mathcal{N}_{\text{best}}$ with lowest $\|W^{(1)\star}\|_1 + \|W^{(2)\star}\|_1$ {"Weight decay"}
  else
    Select $(W^{(1)\star}, W^{(2)\star})$ uniformly at random from $\mathcal{N}_{\text{best}}$
  end if
  Update: $W^{(1)} \leftarrow W^{(1)\star}$, $W^{(2)} \leftarrow W^{(2)\star}$
  Update: $b1 \leftarrow b1(W^{(1)})$, $b2 \leftarrow b2(\beta)$
end for

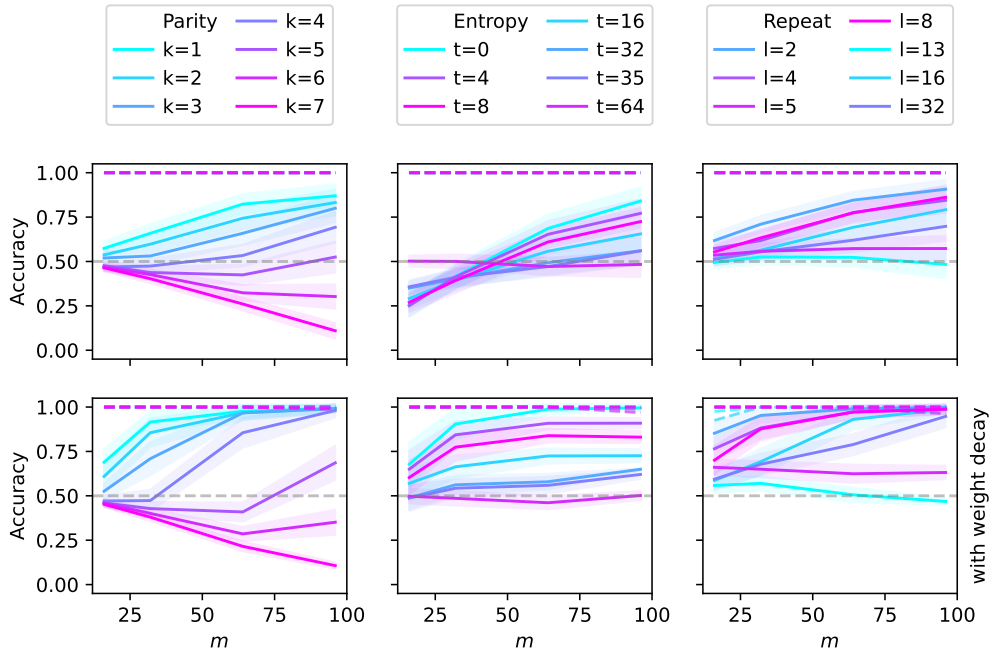


Figure C.7: SGD-like algorithm trained on different targets from the $n = 7$ dataset. Each column shows a different function class – parity, entropy and repeat. See Appendix C.7.1 for full experimental details and a description of each function type. Comparing the top row (no weight decay) and the bottom row (weight decay; for this algorithm, $p = 0.3$) shows that weight decay massively outperforms non-weight decay on targets which have a small minimum representation (read: simple) and makes little difference for functions with a large minimum representation (read: complex functions). $l = 7$ parity is the most interesting example: it has the largest possible minimum representation and thus is the most complex function: the model is biased strongly against it, and thus the more data we give it, the worse it will be.

C.7.5 Relating Algorithm 4 (steepest descent) to Algorithm 2 (Metropolis-Hastings)

Algorithm 2 and Algorithm 4 differ in two key ways.

Probabilistic vs. deterministic greedy acceptance. In the MCMC algorithm, we sample with probability α, which is not the case for the steepest descent algorithm where we always evaluate all 1-hop neighbours in each batch and identify the subset $\mathcal{N}_{\text{best}}$ achieving minimal loss (i.e. highest accuracy), from which the samples are drawn for the next update. We can achieve the steepest descent behaviour in the MCMC algorithm by setting $\kappa \to \infty$ (so that moves reducing the loss are always accepted), but the minimum norm selection then becomes difficult to replicate.

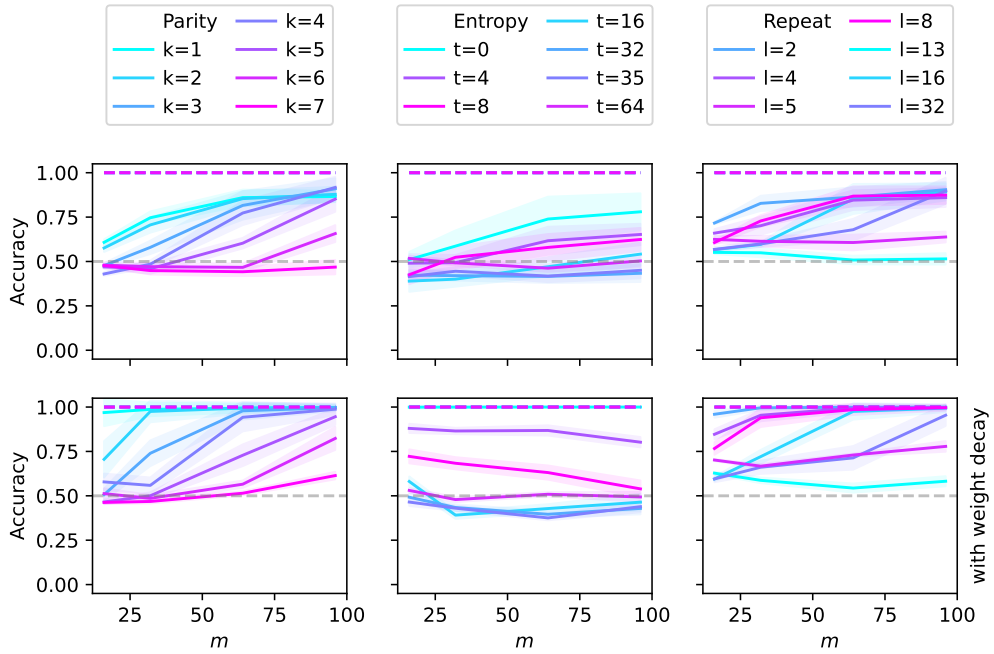


Figure C.8: SGD on continuous network trained on different targets from the $n = 7$ dataset. Each column shows a different function class – parity, entropy and repeat. See Appendix C.7.1 for full experimental details and descriptions of each function type.

Global vs. batch-wise local optimisation The MCMC algorithm is a reversible Markov chain guaranteed to explore the full posterior. On the other hand, the steepest-descent algorithm never revisits rejected states – it hill climbs on randomly selected batch samples. The bias p and norm-based tie-breaking act as heuristic "annealing" and "weight decay," but without detailed balance or stationary-distribution guarantees.

C.7.6 Continuous networks

We also trained 2-layer FCNs with ReLU activations with the same width as the DFCNs. We used SGD with a batch size of $m/2$, and a fixed learning rate of 10^{-3}. The results are shown in Fig. C.8. The only significant difference between the DFCNs trained with either Algorithm 2 or Algorithm 4 is large k-parity, which continuous neural networks have an easier time learning. This is because continuous networks exhibit slightly less inductive bias towards simple functions than DFCNs, which are manually crafted to fully take advantage of this bias for Boolean data.

D
On Complexity

This appendix provides a detailed analysis of the computable complexity measures used throughout this work. We examine two distinct types of measures: the string-based Lempel-Ziv complexity ($K_{LZ}(f)$) and a family of measures based on Disjunctive Normal Form (DNF). In addition to the primary DNF complexity ($K_{DNF}(f)$) used in Chapters 3 and 4, we introduce two new related measures, $K_\theta(f)$ and $K_C(f)$. We analyse the key characteristics of each measure, focusing on their theoretical and empirical scaling properties, which motivates a method for rescaling them. Finally, we connect these practical measures to broader concepts like Zipf's law and ideal learning (Section 2.3), arguing that they serve as effective, interpretable proxies for reasoning about the inductive biases of learning systems.

D.1 Lempel-Ziv complexity

The Lempel–Ziv parsing [245] algorithm proceeds by scanning a finite string x over any alphabet from left to right, maintaining a dictionary of distinct substrings (or "words") observed so far. Whenever the algorithm encounters a new substring that is not already in the dictionary, it records that substring as a new dictionary entry; upon completion of the parse, the dictionary size $N_w(x)$ provides a raw measure of complexity. Intuitively, strings composed of a small number of repeated subpatterns

yield small dictionaries and thus low complexity, whereas strings exhibiting many novel substrings produce large dictionaries and high complexity. We use the definition of LZ-complexity from [65] (see Supplementary Note 7)

$$K_{LZ}(x) \;=\; \frac{\log_2(n)}{2}\Big[N_w(x_{1\ldots n}) + N_w(x_{n\ldots 1})\Big], \tag{D.1}$$

i.e. the average of the forward and reverse parses, which increases the number of distinct complexity values assignable to strings of a given length.

The Lempel–Ziv complexity, $K_{LZ}(x)$, satisfies a number of important asymptotic and finite-size scaling laws. For an ergodic source and in the limit $n \to \infty$, one has

$$\lim_{n\to\infty} \frac{N_w(x)\,\log_2 n}{n} \;=\; h(x), \tag{D.2}$$

where $h(x)$ is the Shannon entropy rate of the source, and consequently $K_{LZ}(x)/n \to h(x)$ for almost all long strings [65]. Complexity is bounded above by entropy, so strings of low entropy cannot exhibit high $K_{LZ}(x)$, while high-entropy strings may nonetheless have simple structure and thus low $K_{LZ}(x)$. Empirically, for short to moderate n, $K_{LZ}(x)$ often outperforms generic lossless compressors in approximating Kolmogorov complexity [65]. As n increases, the mean and median of the normalised complexity distribution approach unity, and the relative standard deviation σ/μ decreases, indicating that typical complexities concentrate sharply around their mean. Strings whose complexity lies well below the mean become exponentially rare in n, although a small fraction of "maximally complex" strings arise anomalously via simple LZ-specific constructions. Additive and multiplicative constants in $K_{LZ}(x)$ may be absorbed into fitting parameters when modelling such simplicity-bias phenomena.

Eq. (D.2) shows us that $K_{LZ}(f)$ of a random function (with $h(x) = 1$) scales as $O(n)$ (as will the complexity of the maximum complexity function).

D.2 DNF Complexity Measures

Each nonzero entry in $W^{(1)}$ corresponds bijectively to a literal in some conjunctive clause of the DNF (Proposition 3.1.1). Hence

$$\|W^{(1)}\|_1 \;=\; \sum_{i,j} \mathbb{1}[W^{(1)}_{ij} \neq 0] \;=\; L(\Phi_f). \tag{D.3}$$

Minimising this quantity gives us the complexity measure used in the main text, $K_{DNF}(f)$ (Proposition 3.1.2). However, strictly speaking, this is not the true norm of the DFCN, which must take into account the second layer: $\|\theta\|_1 := \|W^{(1)}\|_1 + \|W^{(2)}\|_1$. We define a second type of DNF-related complexity

$$K_\theta(f) = \min_{W^{(1)},W^{(2)}} \left(\|W^{(1)}\|_1 + \|W^{(2)}\|_1\right) \tag{D.4}$$

Using this norm is equivalent to defining the DNF complexity as the number of literals plus the number of clauses in the minimum representation.

Relating $K_{DNF}(f)$ to $K_\theta(f)$ We can lower bound the number of clauses as a function of the number of literals by creating $\lceil K_{DNF}(f)/n \rceil$ unique clauses with at most n elements. We can upper bound this by remembering that if clauses span k columns in $W^{(1)}$, we can have no more than 2^{k-1} clauses in the minimum representation. This means we can never have more than $2^{\lceil k \rceil - 1}$ clauses, where k satisfies $K_{DNF}(f) = k2^{k-1}$. We can rearrange and take logs to obtain

$$K_{DNF}(f) + \lceil K_{DNF}(f)/n \rceil \leq K_\theta(f) \leq K_{DNF}(f) + 2^{\lceil 1+\log_2 K_{DNF}(f) \rceil - 1}. \tag{D.5}$$

The maximum of each complexity is parity, $K_{DNF}(f) = \frac{n}{2}2^n$ and $K_\theta(f) = \frac{n+1}{2}2^n$.

Problems with $K_{DNF}(f)$ and $K_\theta(f)$ The maximum complexity for these two measures is $O(n2^n)$: ideally, the maximum function should not have complexity greater than $2^n + O(1)$. Assuming complexity is related to compression – that is, simple functions are highly compressible – we should not "compress" a function to more bits than its string representation, which needs 2^n bits. One complexity measure that does satisfy this requirement is double the total number of clauses

$$K_C(f) = 2\|W^{(2)}\|_1, \tag{D.6}$$

which has a maximum complexity of exactly 2^n. $\lceil K_{DNF}(f)/n \rceil$ provides the minimum number of clauses we can fit $K_{DNF}(f)$ literals in, and we can again use the fact that if the maximum clause has length k, we can have no more than 2^{k-1} clauses in the minimum representation to upper bound,

$$\lceil K_{DNF}(f)/n \rceil \leq K_C(f) \leq 2^{\lceil 1+\log_2 K_{DNF}(f) \rceil - 1}. \tag{D.7}$$

D.3 Rescaling complexity measures

We can make some of these observations precise using known results. Blais and Tan [246] lists the most important. The Korshunov-Kuznetsov Theorem states that a random boolean function requires $\Theta(2^n/\log n \log\log n)$ clauses, each with an expected $n - \Theta(\log n + \log\log n)$ literals. From this, we have the following scaling laws.

Complexity	constant	t-entropy	k-parity	Random	Parity
$K_{DNF}(f)$	$O(1)$	$O(nt)$	$k2^{k-1}$	$\Theta\left(\frac{n2^n}{\log n \log\log n}\right)$	$n2^{n-1}$
$K_\theta(f)$	$O(1)$	$O\left((n+1)\,t\right)$	$(k+1)2^{k-1}$	$\Theta\left(\frac{(n+1)2^n}{\log n \log\log n}\right)$	$(n+1)2^{n-1}$
$K_C(f)$	$O(1)$	$O(2t)$	2^k	$\Theta\left(\frac{2^n}{\log n \log\log n}\right)$	2^n
$K_{LZ}(f)$	$O(1)$	$O(2t)$		$\Theta(2^n)$	

We should be able to use the result for a random function to rescale each complexity measure to behave optimally – that is, the number of functions grows with 2^K. $K_{LZ}(f)$ has already been rescaled by Dingle et al. [167] in its definition (Eq. (D.1)), leaving

$$K'_{DNF} = \frac{\log n}{n} K_{DNF}(f) \quad K'_\theta = \frac{\log n}{n+1} K_\theta(f) \quad K'_C = (\log n) K_C(f),$$

where we ignore the $\log\log n$ factors for clarity. Under these rescaled measures, a random Boolean function on n variables will have complexity $\Theta(2^n)$.

At the comparatively small scales we are able to study ($n = 7$), we cannot see this asymptotic behaviour. Fig. D.1 shows that the DNF complexity $K_{DNF}(f)$ of a string of length 2^n is roughly equal to 2^n (as is $K_{LZ}(f)$, consistent with results in [64]). The complexity of the parity function scales sub-linearly in length for

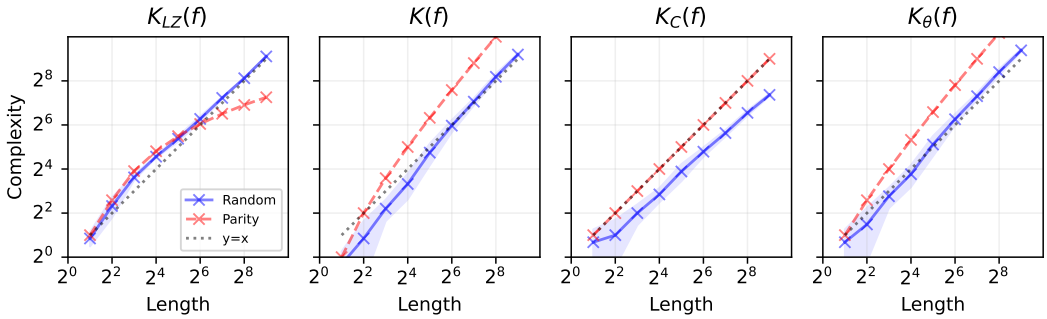


Figure D.1: Scaling of $K_{LZ}(f), K_{DNF}(f), K_C(f), K_\theta(f)$ for random Boolean functions and the parity function for $n = 2$ to $n = 9$ (the length of the string representation of f is therefore 2^n). [Note that $K = K_{DNF}$ in this figure]. We expect random functions to be incompressible, and thus have a complexity $\approx 2^n$ for good complexity measures. LZ complexity is known to satisfy this requirement up to $O(1)$ terms [245], and for $K_C(f)$ the worst case is exactly 2^n. For $K_{DNF}(f), K_\theta(f)$ however, the worst case (parity) is $\frac{n}{2}2^n$ and $\frac{n+1}{2}2^n$, respectively, and whilst the typical random functions appear to have complexities close to 2^n for small n, theoretical results in Appendix D.2 show that this would change as n increases.

$K_{LZ}(f)$, but has $K_{DNF}(f) = n2^{n-1}$. For $K_C(f)$, parity has complexity exactly 2^n, with a random function slightly less. $K_\theta(f)$ behaves like $K_{DNF}(f)$.

Fig. D.2 shows distributions over complexity from uniformly sampling a binary string ($n = 7$, so length 128) in the first column. We argued in Chapter 4 that for uniform sampling of binary strings, $P(K) \sim 2^K$ – see Fig. 4.1(g). We also argued that for a well-conditioned learning agent, $P(K) = O(1)$, and we should see Zipf's law in the prior (c.f. Appendix D.4.

Fig. C.3 shows that the continuous NN and Discrete NN both have Zipfian priors. This means $P(f) = 2^{K_R(f)}$, where $K_R(f) = \log_2 \text{rank}$. In this, the rank acts as its own "internal complexity". $K_{LZ}(f)$ behaves as we would expect for uniform sampling (see Fig. 4.1(b) for more details) and for the Continuous NN. $K_{DNF}(f)$ is qualitatively similar for Uniform sampling and the Continuous NN, but we see a much shorter plateau than with $K_{LZ}(f)$. For the discrete NN, both measures fall short, dropping off too fast. $K_C(f)$ does not behave as we would expect at this scale. However, this is just $n = 7$ – the correct asymptotics may well appear for larger n outside our ability to sample.

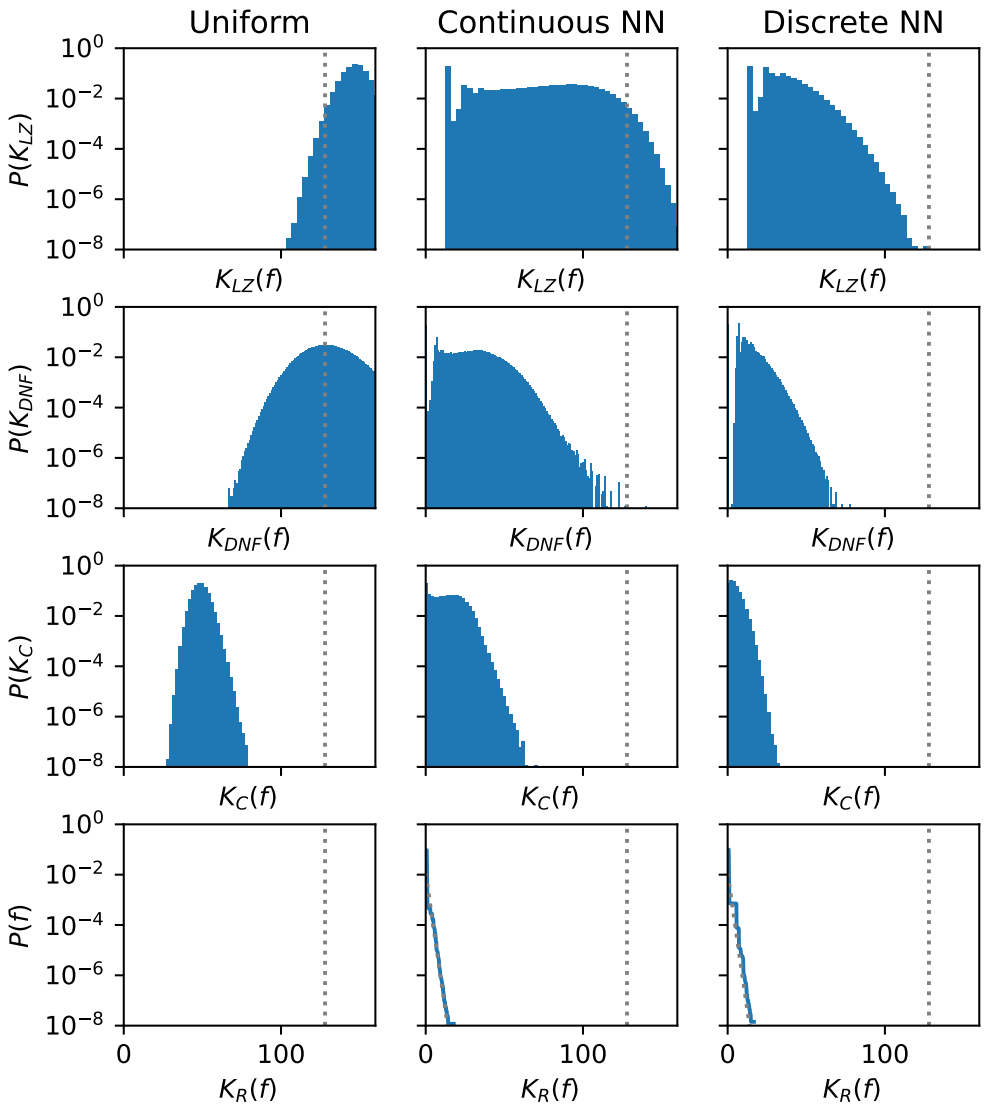


Figure D.2: Complexity vs probability for $n = 7$. Grey dotted line at $K = 128$. The first column shows the result of uniformly sampling strings (i.e. randomly generating strings). The second column shows the prior of a 1–layer FCN on the Boolean data, and the third the prior of the discrete NN. Here $K = K_{\text{DNF}}$ and $P(K)$ is the probability of generating a function of complexity K for three different complexity measures. Note that the diagonal dotted grey lines in the bottom row show the Zipf's law fit from Appendix D.4.

D.4 Zipf's law

Valle-Pérez et al. [64] showed that the prior of neural networks is well-described by Zipf's law. For a dataset of size 2^n, the probability of randomly initialising to a function $P(f)$ is a function of the rank of the function $R(f)$ (where the rank of the most probable function is 1, the second most is 2 and so on) that satisfies

$$P(f) = \frac{1}{2^n \log 2} \frac{1}{R(f)}, \tag{D.8}$$

where the first factor is a normalisation term given that there are 2^{2^n} functions. We show in Chapter 5 that when $P(f)$ satisfies Zipf's law, a Bayesian learning agent on this prior will learn optimally. We find it a useful reference point when considering the scaling laws in Table 3.2.

Note that when $P(f)$ satisfies Eq. (D.8), we can write $K_R(f) = \log_2 R(f)$, and so $P(f) \propto 2^{-K_R(f)}$.

D.5 What are the takeaways?

This appendix aimed to identify and validate complexity measures for analysing the inductive biases of learning systems. An effective measure for this purpose should exhibit two key properties

1. **interpretability,** where the complexity value corresponds to a tangible property of the function (e.g., its logical structure)

2. **AIT correspondence,** where the measure aligns with the principles of algorithmic probability, such that the prior probability of a function $P(f)$ is inversely related to its complexity $K(f)$, ideally satisfying a Levin-like relation $P(f) \propto 2^{-K(f)}$.

A crucial conclusion is that no single, universal complexity ranking exists across all learning systems. While one can define a complexity $K_R(f) = \log_2 R(f)$ from the empirically observed probability rank $R(f)$ of a function under a given model, this measure is both uninterpretable and architecture-specific. Different network

architectures will invariably produce different function rankings, even if they all exhibit Zipfian priors. This is analogous to the fact that different UTMs U and V have different Kolmogorov complexities $K_U(f)$ and $K_V(f)$, which are related by $K_U(x) \leq K_V(x) + c$ for some constant c.

A pragmatic compromise is required between measures that are interpretable within the DFCN framework and those that possess theoretically desirable scaling properties. On one hand, the DNF-based measures ($K_{DNF}(f)$, $K_\theta(f)$, and $K_C(f)$) are highly interpretable and intrinsically linked to the structure of the networks we study (i.e. the DFCNs). However, some of these, particularly $K_{DNF}(f)$ and $K_\theta(f)$, exhibit poor asymptotic scaling. On the other hand, Lempel-Ziv complexity ($K_{LZ}(f)$) demonstrates the appropriate average scaling of $\Theta(2^n)$ for a random function, but as a generic string-compression algorithm, it is not directly tied to the specific architectural biases of a DFCN. The most effective path forward may therefore be to improve the DNF-based measures by rescaling them, which corrects their asymptotic behaviour while retaining their valuable connection to the model's structure.

Although these interpretable measures do not perfectly represent the network's intrinsic bias, they serve as important effective **proxies** for us to reason over.

E

Appendices for Chapter 4

E.1 Definitions & Algorithms

In Appendix E.1 we set out the notation for this section, including the boolean system and neural networks

E.1.1 The boolean system

The n-dimensional boolean dataset $\mathcal{B}(f)_n = \{(x_1, f(x_1)) \dots (x_{2^n}, f(x_{2^n}))\}$, where x_i covers $\{0,1\}^n$ and $f(x_i) \in \{0,1\}$. This means the dataset is completely described by a function

$$f : \{0,1\}^n \to \{0,1\}.$$

This means f can be represented as a binary string of length 2^n, by ordering inputs ascending by their binary value, and concatenating their outputs y. This is equivalent to assigning 0 or 1 to every corner of the n-dimensional boolean hypercube, and this is the form the DNN receives. There are therefore 2^{2^n} different functions f.

For example, for the $n = 2$ and $g = 0110$, $\mathcal{B}(g)_2 = \{(00, 0), (01, 1), (10, 1), (11, 0)\}$. The DNN, which requires inputs in $\mathbb{R}^2$ and outputs in $\mathbb{R}$ will receive the following input-target pairs: $\mathcal{B}(g)_2 = \{([0,0], 0), ([0,1], 1), ([1,0], 1), ([1,1], 0)\}$.

The LZ complexity of f is therefore the complexity of its string representation. This representation has the ordering property built in – the string can be decoded

into the (x_i, y_i) pairs using $\mathcal{B}(f)_n = \{(bin(i), f[i])\}_{i=0}^{i=2^n}$ where $f[i]$ denotes the i'th digit in f, and $bin(i)$ is the binary representation of i. The complexity of the dataset is therefore $K(\mathcal{B}(f)_n) \leq K_{ordering} + LZ(f)$, where $K_{ordering}$ is the complexity of the decoding process. Using a much more complex ordering would change the values of LZ complexity measured (and lead to very different distributions over $P(K)$) but fundamentally not capture the complexity of the dataset, as $K_{ordering}$ would be much larger.

Training set S_m. A training set with m elements, for a dataset $\mathcal{B}(g)_n$, $S(g)_m = \{(x_i, g(x)_i)\}_{i \in s(g)_m}$ where $s(g)_m$ is an index set of size m. Normally, the g is left implicit, so $S(g)_m$ is denoted S_m. There are $\binom{m}{2^n}$ unique training sets of size m for each function g, and $S_{2^n} = \mathcal{B}(g)_n$.

E.1.2 Deep Neural Network notation

Fully connected networks are a family of models parameterised by θ of the form

$$f : \mathbb{R}^I \to \mathbb{R}^O \tag{E.1}$$

$$x \to f(x; \theta). \tag{E.2}$$

Fully connected networks are defined recursively by

$$x_i^{(l)} = \sigma(z_i^{(l)}) = \sigma\left(\sum_{j=1}^{j=w^{(l)}} W_{ij}^{(l)} x_j^{(l-1)} + b_j^{(l)}\right) \quad \text{for } 1 \leq l < L \tag{E.3}$$

$$N(x_0; \theta) = g(z_i^{(L)}) \tag{E.4}$$

where $W_{ij}^{(l)}$ and $b_j^{(l)}$ are the weights and biases respectively, parameterised by θ. x_0 is the input to the first layer (normally just denoted x), σ is an activation function, and g is a likelihood function, which is the identity for mean square error (MSE) loss and softmax for cross-entropy (CE) loss.

We restrict the DNN N to the domain $x \in \{0,1\}^n$ and codomain $\{0,1\}$ by defining $f_N(x)$

$$f_N(x) = \mathbb{1}(N(x; \theta) > t)$$

where $t = 0$ (except when DNNs are trained with MSE loss, where 0.5 is used).

For an n-dimensional boolean system, a neural network with input dimension n and output dimension 1 has the correct dimensions to model $\mathcal{B}_n$. Throughout the main text, we use $n = 7$, with fully connected DNNs with $1 \leq N_L \leq 10$ hidden layers with width 40. Weights are initialised i.i.d. from $\mathcal{N}(0, \sigma_w)$ for $1 \leq \sigma_w \leq 8$, and biases are initialised from $\mathcal{N}(0, \sigma_b)$ where $\sigma_b = 0.02, 0.2$. We found that these differences did not induce substantial differences in network behaviour, particularly after training. The DNNs use tanh activations, unless otherwise specified.

E.1.3 Prior over functions, P(f)

We define the prior distribution over functions $P(f) = P(f; N, \sigma_w, \sigma_b)$ with respect to a DNN N with input dimension n and parameter initialisation σ_w, σ_b, $\mathcal{B}(f)_n$

$$P(f) = \int_\theta d\theta P(\theta) \prod_{j=0}^{j=2^n-1} \mathbb{1}(f_N(x_i; \theta) = f(x_i)) \tag{E.5}$$

$$= \int_\theta d\theta P(\theta) \mathbb{1}(\mathcal{B}(f_N(\cdot; \theta))_n = f) \tag{E.6}$$

where $P(\theta)$ is the joint distribution over weights and biases sampled i.i.d. from $\mathcal{N}(0, \sigma_w)$ and $\mathcal{N}(0, \sigma_b)$. $P(f)$ is the prior distribution over functions. We approximate the integral by randomly initialising parameters 10^8 times, record the frequencies ρ_i of observed f_i, and use $P(f_i) \approx \rho_i / 10^8$. In all our experiments $\sigma_b = 0.1\sigma_w$, but any relation (or none at all) is perfectly acceptable – we used this relation because trained neural networks typically end up training to small values of σ_b.

E.1.4 Prior over LZ complexity, P(K)

The prior distribution over complexities K

$$P(K) = \sum_{f: LZ(f) = K} \int_\theta d\theta P(\theta) \mathbb{1}(\mathcal{B}(f_N(\cdot; \theta))_n = f) \tag{E.7}$$

where $LZ(f)$ is the LZ complexity of the string representation of f. $P(K)$ is approximated by randomly initialising parameters 10^8 times, recording the frequencies ρ_i of observed f_i, so $P(K) \approx \sum_{f_i: LZ(f_i) = K} \rho_i / 10^8$.

E.1.5 Obtaining functions with a range of LZ complexity

This method is designed to generate functions with a range of LZ complexities, without using a DNN (because the other method: sampling from the prior of a DNN with $\sigma_w = 1$ might introduce a bias towards specific types of functions LZ complexity is not able to capture, which would bias results).

Algorithm 5 Generating strings over a range of complexities

input: Length of binary string, 2^n (typically $n = 7$). $F = []$.
for m in $0, 1, \ldots, 2^{n-1}, 2^n$ **do**
  for _ in range 10^6 **do**
    Generate a string s with the first m characters 1 and all others 0. Generate a random permutation S_{2^n}, and use it to permute s.
    Append the tuple $(s, LZ(s))$ to F if $s \notin F$.
  end for
end for
group functions in F by K.
Return F

E.1.6 Generalisation error vs LZ complexity

This method is designed to determine the expected generalisation error for DNNs with L layers, tanh activations, and initialisation scheme (σ_w, σ_b), for some training set size m and for a range of target function complexities.

Algorithm 6 Generalisation error vs LZ Complexity

input: DNN N with L layers tanh activations, initialisation variances σ_w ($\sigma_b = 0$), optimiser O, batch size b. Dataset $\mathcal{B}(\cdot)_7$, training data size $0 < m < 2^7$ (typically $m = 64$). Set F of functions generated by Algorithm 5. Stochastic optimiser O, Loss function L.
Let $F_{out} =$
for unique K in F **do**
  Sample one function f_K
  $C_K = []$
  for _ in range(1000) **do**
    Randomly sample m integers without replacement from $0, \ldots, 2^7$. These are the indices of f_K used as training data S_m. The other part of $B(\cdot)_7$ is the test set E
    Train DNN with optimiser O with loss function L on S_m until 100% training accuracy is achieved. Record the error on E, ϵ
    Append ϵ to C_K
  end for
  Calculate mean μ_K and std ρ_K of C_K. Append (K, μ_K, ρ_K) to F_{out}
end for
Return F_{out}.

For experiments with CE loss, the AdvSGD optimiser was used (see Appendix A in [64] for details) was used with batch size 16. For experiments with MSE loss, the Adam optimiser was used, with learning rate 10^{-5} and batch size 16.

E.1.7 Generalisation vs output function LZ complexity

These experiments aim to capture the distribution of functions found when training a DNN to some target f with training set size m.

Algorithm 7 $\epsilon(f_N)$ vs $LZ(f_N)$ for target function f

input: DNN N with L layers tanh activations, initialisation variances σ_w ($\sigma_b = 0$), optimiser O, batch size b. Dataset $\mathcal{B}(\cdot)_7$, training data size $0 < m < 2^7$ (typically $m = 64$). Set F of functions generated by Algorithm 5. Stochastic optimiser O, Loss function L. Target complexity K.
Let $F_{out} =$
Sample one function f_K from F
for _ in range(1000) **do**
  Randomly sample m integers without replacement from $0, \ldots, 2^7$. These are the indices of f_K used as training data S_m. The other part of $B(\cdot)_7$ is the test set E
  Train DNN with optimiser O with loss function L on S_m until 100% training accuracy is achieved. Record the error on E, ϵ, and $LZ(f_N(\theta)$.
  Append $(K, \epsilon, LZ(f_N(\theta))$ to F_{out}
end for
Return F_{out}.

E.1.8 Likelihood estimation and Approximate Bayes

Our starting point is Bayes' rule

$$P(f \mid S) = P(S \mid f)P(f)/P(S) \tag{E.8}$$

where $P(S) = \sum_f P(S \mid f)P(f)$. For a dataset $\mathcal{B}(g)_n$, using $S_m = S(g)_m$, we want to calculate

$$\langle P(f \mid S_m)\rangle = \binom{2^n}{m}^{-1} \sum_{S_m \in \mathcal{S}_m} P(f \mid S_m) = \binom{2^n}{m}^{-1} \sum_{S_m \in \mathcal{S}_m} P(S_m \mid f)P(f)/P(S_m) \tag{E.9}$$

Where $\mathcal{S}_m$ is the set of possible training sets of size m, taken from the target function g. We first assume $P(S_m) \approx \langle P(S_m)\rangle_{S_m \in \mathcal{S}_m}$, and take the likelihood function $P(S_m \mid f) = \mathbb{1}(\mathcal{B}(f)_n \mid_{S_m} = S_m)$. Then

$$\langle P(f \mid S_m)\rangle_{S_m \in \mathcal{S}_m} \approx \frac{1}{\langle P(S_m)\langle_{S_m \in \mathcal{S}_m}} \binom{2^n}{m}^{-1} \sum_{S_m \in \mathcal{S}_m} P(S_m \mid f)P(f) \tag{E.10}$$

$$= \frac{1}{\langle P(S_m)\rangle_{S_m \in \mathcal{S}_m}} \binom{2^n}{m}^{-1} \binom{m}{2^n(1-\epsilon_g(f))} (1-\epsilon_g(f))^m P(f) \tag{E.11}$$

$$:= E(f)P(f) \tag{E.12}$$

where $\binom{m}{2^n(1-\epsilon_g(f))} = 0$ if $m > 2^n(1-\epsilon_g(f))$ and $\epsilon_g(f)$ is the fractional error of g with respect to f (usually just denoted $\epsilon(f)$). We then use this approximation for $\langle P(f \mid S_m)\rangle_{S_m \in \mathcal{S}_m}$ in

$$\langle P(K \mid S_m)\rangle_{S_m \in \mathcal{S}_m} = \sum_{f:LZ(f)=K} \langle P(f \mid S_m)\rangle_{S_m \in \mathcal{S}_m} = \sum_{f:LZ(f)=K} E(f)P(f) \tag{E.13}$$

$$= P(K) \sum_{f:LZ(f)=K} E(f)P(f)/P(K) \tag{E.14}$$

$$\approx P(K) \frac{1}{|\{f : LZ(f) = K\}|} \sum_{f:LZ(f)=K} E(f) \tag{E.15}$$

if we assume that for all f with $LZ(f) = K$, $P(f)$ is roughly equal. Finally, if we assume that our method for sampling f (Algorithm 8) to produce a distribution over $E(f)$ samples functions f in a unbiased manner, then we have

$$\langle P(K \mid S_m)\rangle_{S_m \in \mathcal{S}_m} \approx P(K) \sum_{f:LZ(f)=K} E(f)P(f)/P(K) \tag{E.16}$$

$$\approx \sum_{f:LZ(f)=K} \frac{1}{|\{f : LZ(f) = K\}|} \sum_{f:LZ(f)=K} E(f) \tag{E.17}$$

$$\propto P(K) \frac{1}{|\{f : LZ(f) = K\}|} \text{top k}(E(f_K)) \times \frac{\binom{2^n}{2^n\epsilon_g(f)}}{\binom{m}{2^n(1-\epsilon_g(f))}} \tag{E.18}$$

where top k$(E(f))$ refers to the sum of the k largest $E(f_K)$ found for f with $LZ(f) = K$. This assumes that most of the weight in $\sum_{f:LZ(f)=K} E(f)$ comes from a small number of the largest $E(f)$. Each $E(f_k)$ is divided by $\binom{m}{2^n(1-\epsilon_g(f))}$ to account for that number of functions with the same error but distributed differently.

Algorithm 8 Approximate Bayes

input: Dimension n, Prior $P(K; N, \sigma_w, \sigma_b)$, target function f, integer k (typically 5)
Part 1: Estimating $P(S_m \mid K)$
$F = []$
for b in [8,16,32,64,100] **do**
  for b' in range(10000) **do**
    Perform $(b'\%b)$ bit flips of f, let this function be denoted g. If g has not already been found, record the expected generalisation error averaging over test sets
    $P(g) = \binom{2^n}{m}(1-\epsilon(g))^m$ where $\epsilon(g) = \sum_i \mathbb{1}(f[i] \neq [i])$.
    Append $(LZ(g), P(g))$ to F
  end for
end for
$P(S_m \mid K) = []$
for Complexities K within F_{out} **do**
  $P_k(K) = (1/k) \sum_{\text{top } k \; g:LZ(g)=K} P(g)$
  Append $(K, P_k(K))$ to $P(S_m \mid K)$
end for
Return $\langle P(K \mid S_m) \rangle_{S_m} \approx P(K)P(K; N, \sigma_w, \sigma_b)$

Note that this approximation for the posterior distribution with 0-1 likelihood

$$\langle P(f \mid S_m) \rangle_{S_m} = \binom{2^n}{m}^{-1} \sum_{S_m \in \mathcal{S}_m} \mathbb{1}(f \text{ consistent with } S_m) P(f)/P(S_m) \tag{E.19}$$

is not the posterior distribution DNNs may approximate. Gaussian Processes with MSE use a mean-square error likelihood $P(S_m \mid f) = \exp\left(-\sum_{i \in s_m} (f(x_i;\theta) - g(x_i))^2 / 2\alpha^2\right)$, where α^2 is typically very small. Despite halting training at 0 training error, we find that the DNNs often also arrive at very low training loss.

E.1.9 Bias-Variance Tradeoff Experiments

These experiments evaluate the effects of changing the size of a learning agent's hypothesis class $\mathcal{H}$ (a hypothesis class is the set of functions a learning agent may consider a candidate function to describe the data). For the boolean system, we can defined hypothesis classes with a maximum LZ complexity

$$\mathcal{H}_k = \{f : K(f) < k\}$$

meaning the hypothesis class of all functions with LZ complexity less than k.

The uniform learner with dataset $\mathcal{B}(g)_n$ and training set $S(g)_m$, hypothesis class $\mathcal{H}_k$ generates a distribution over hypotheses

$$P_k(f \mid S(g)_m) = \frac{\mathbb{1}(\epsilon_T(f) = \min\{\epsilon_T(f') : f' \in \mathcal{H}_k\})}{\sum_{f^*} \mathbb{1}(\epsilon_T(f^*) = \min\{\epsilon_T(f') : f' \in \mathcal{H}_k\})} \tag{E.20}$$

In other words, functions achieving the best possible training error are given equal weighting within $\mathcal{H}_k$. For $n = 5$ systems with only $2^32 \sim 10^9$ strings for $\mathcal{H}_\infty$, this process can be done exhaustively for any fixed training set of any size. For the $n = 7$ system, this is done with $m = 100$, meaning exact values can be obtained for $\mathcal{H}_k$ sufficiently expressive to fit the training data, by exhaustively enumerating over all 2^28 strings that fit the data. When $\mathcal{H}_k$ is not sufficiently expressive, 10^6 strings are sampled by perturbing k (for a uniformly sampled integer k in range $0 < k < 10$) labels using the a) true function b) all 0's or all 1's function and c) a randomly generated function with high symmetry, as bases for the perturbation. This generated a distribution over low complexity functions with low train error, with which we calculated $P(f \mid S(g)_m)$ for k less than the minimum k capable of fitting the training set. This process produced results that looked qualitatively similar to those seen for the $n = 5$ system (which is exact).

The neural networks with dataset $\mathcal{B}(g)_n$ and training set $S(g)_m$, hypothesis class $\mathcal{H}_k$ generates a distribution over hypotheses

$$P_k^{(NN)}(f \mid S(g)_m) = \frac{P_{SGD}(f \mid S(g)_m)\mathbb{1}(f \in \mathcal{H}_k)}{\sum_{f'} P_{SGD}(f' \mid S(g)_m)\mathbb{1}(f' \in \mathcal{H}_k)}.$$

This distribution is approximated by training 5000 DNNs to 100% training accuracy, and rejecting all functions not found in each $\mathcal{H}_k$, as with the random learner.

E.1.10 The Gaussian Processes/Linearised Approximation

We use a very wide neural network and train only the last layer with MSE loss until low loss for a linear model closely approximating a Gaussian Process with mean square error likelihood – see [2, 247] for an explanation of Gaussian Processes with MSE likelihood and how they are equivalent to these linear models respectively. We use DNNs with width 16384, freeze all but the final classification layer, and train this layer with Adam (without weight decay) and MSE loss until the loss reaches 10^{-5}.

E.1.11 Experimental details for image datasets & CSR

We also use MNIST dataset [116] and CIFAR10 dataset [154]. Because functions on the datasets do not have an easily calculable intrinsic measure (not dependent on the neural network) of complexity, we use CSR.

For both MNIST and CIFAR10, we use DNNs with hidden layer width 200, input width 784 and 3072 respectively and output dimension 1.

The critical sample ratio (CSR), a measure of the complexity of a function, was introduced in [248]. It is defined with respect to a sample of inputs as the fraction of those samples which are critical samples, defined to be an input such that there is another input within a box of side $2r$ centred around the input, producing a different output (for discrete outputs). Following [64], we use CSR as a rough estimate of the complexity of functions found in the posterior (conditioning on S), and the prior (i.e. functions on S). In our experiments, we used binarised MNIST with a training set size of 10000 and a test set of size 100 (analogously to the majority of our other experiments).

E.1.12 Infinite depth chaotic tanh-activated DNNs have a uniform prior

Signal propagation in deep chaotic networks was extensively studied in [79, 80]. The most relevant result for our purposes concerns the correlations between activations at l layers, for neural networks at initialisation, taking the mean field approximation (where preactivations z_i^l are replaced by a Gaussian whose first two moments match those of z_i^l). This mean field approximation is exact for infinitely wide neural networks.

Specifically, it is shown in [79] that for neural networks with tanh activations and combination of σ_w and σ_b in the "chaotic regime" (for example, $\sigma_w = 2.5$, $\sigma_b = 0.0005$), as the number of layers tends to infinity

$$\mathbb{E}[z_i^l(x_a) z_j^l(x_b)] \xrightarrow{l\to\infty} q^* \delta_{ij}(\delta_{ab} + c_{ij}(\sigma_w, \sigma_b))) \tag{E.21}$$

where x_a and x_b are inputs to the network, and z_i^l is the i'th preactivation in the l'th layer. $c_{ij}(\sigma_w, \sigma_b)$ is a correlation term between inputs i and j, and it is further shown that $(\sigma_w, \sigma_b) \xrightarrow{\sigma_w \to \infty} 0$ for any fixed σ_b. By contrast, neural networks in the ordered regime (characterised by a small σ_w and large σ_b) have an attractive fixed point at 1 – all inputs become perfectly correlated.

For a neural network with $L \to \infty$ infinitely wide hidden layers and a single output neuron, Eq. (E.21) can be used with $i = j = 1$, and $f(x_a)$ the final layer's output for input x_a to give

$$K(x_a, x_b) = \mathbb{E}[f(x_a)f(x_b)] \xrightarrow{l \to \infty} q^* \delta_{ab},$$

which defines the kernel used in Neural Network Gaussian Processes (NNGPs). The prior probability of an output y, $P(y)$ is given by

$$P(y) \propto \exp\Big\{-y^T K^{-1} y\Big\},$$

where K is the kernel matrix for m inputs, and y the output vector (pre-thresholded) for the m outputs. As K^{-1} is a diagonal matrix, all predictions are uncorrelated and have equal chance of being 0 or 1, so the probability of any thresholded function f must be uniform:

$$P(f) = 2^{-m}.$$

A similar result was proved in [249] but for NTK – NTK in the limit of infinite depth converges to an equicorrelated matrix with correlation $c = c(\sigma_w, \sigma_b)$, with the same limiting behaviour as the NNGPs.

E.1.13 PAC-Bayes appied to a uniform learner

For the random learner with complexity cutoff (Eq. (E.20)), for the hypothesis class $\mathcal{H}_k$, $P(S) = \frac{|\{f : K(f) < k, f|_S = S\}|}{|\{f : K(f) < k\}|}$. Plugging this into Eq. (A.16),

$$\epsilon(Q^*) \leq 1 - \exp\left(-\frac{\ln \frac{|\{f:K(f)<k\}|}{|\{f:K(f)<k, f|_S=S\}|} + \ln \frac{2m}{\delta}}{m-1}\right). \tag{E.22}$$

If at least one function achieves zero test error, $|\{f : K(f) < k, f|_S = S\}| \geq 1$, so $P(S) > \frac{1}{|\mathcal{H}_k|}$, and can straightforwardly be plugged into Eq. (A.16) – which produces the PAC Bound:

$$\epsilon(Q^*) \leq 1 - \exp\left(-\frac{\ln|\mathcal{H}_k| + \ln\frac{2m}{\delta}}{m-1}\right) \sim 1 - \exp\left(-\frac{k\ln 2 + \ln\frac{2m}{\delta}}{m-1}\right). \tag{E.23}$$

where the second relation holds if we assume that $\mathcal{H}_k \sim 2^k$, Finally, as a sanity check, if $\mathcal{H}_{k_{max}} = 2^m$ (all functions are available), so

$$\epsilon(Q^*) \leq 1 - \frac{1}{2}\underbrace{\exp\left(-\frac{\ln\frac{2m}{\delta}}{m-1}\right)}_{\leq 1}. \tag{E.24}$$

We use Eq. (E.22) for the $n = 5$ system, as all functions can be enumerated. For $n = 7$, we use the bound in Eq. (E.23), with a cutoff at $|H_{k_{max}}| = 2^{2^7}$ (required because the maximum LZ complexity is greater than 128).

E.1.14 The unbiased learner

The unbiased learner $U_{\leq K_M}$ with LZ complexity cutoff is defined in Algorithm 9. We can learn boolean strings by setting a maximum complexity K_M and rejecting all functions with LZ greater than that value. The prior, $P_{\leq K}(f)$, and likelihood rule, $P_{\leq K}(S \mid f)$, can be defined as follows:

$$P_{\leq K}(f) = \mathbb{1}[f \in \mathcal{H}_{K_m}]/|\mathcal{H}_{K_m}|$$
$$P_{\leq K}(S \mid f) = \mathbb{1}\left[\epsilon_S(f) == \min\{\epsilon_S(f') : f' \in \mathcal{H}_{K_m}\}\right]$$

where $\epsilon_S(f)$ denotes the error of a function f on the training set S. Explicitly, we have a uniform prior over functions with $K_{LZ} \leq K_M$, and a posterior distribution that is uniform over functions with $K_{LZ} \leq K_M$ that achieve the minimum possible training error.

Algorithm 9 Unbiased Learner $U_{\leq K_M}$

Input: target function f_t, training set size m, complexity cutoff K_M
Let S be the training set of m input-output pairs from f_t.
Let T be the test set of $2^n - m$ input-output pairs from f_t.
Initialize empty lists for errors: $E_e \leftarrow [\,]$ for each $K \in \{0, \ldots, m\}$.
for each function f in the space of all possible functions **do**
  Calculate the Kolmogorov complexity $K_{LZ}(f)$.
  if $K_{LZ}(f) \leq K_M$ **then**
    Calculate the training error $\epsilon_S(f)$ on the training set S.
    Calculate the generalization error $\epsilon_G(f)$ on the test set T.
    Append $(\epsilon_S(f), \epsilon_G(f))$ to the list $E_{\epsilon_S(f)}$.
  end if
end for
Let $e^* = \min\{e \mid E_e \text{ is non-empty}\}$. The predicted generalization error is the average of the generalization errors for the functions in E_{e^*}:

$$\mathbb{E}[\epsilon_G|\epsilon_S = e^*] = \frac{1}{|E_{e^*}|} \sum_{(\epsilon_S, \epsilon_G) \in E_{e^*}} \epsilon_G$$

PAC-Bayes applied to $U_{\leq K}$ For the unbiased learner $\mathcal{U}_{\leq K}$, target function f_t with training set S of size m, $P(U(S)) = \frac{|\{f : K_{LZ}(f)) \leq K, f|_S\}|}{|\{f : K_{LZ}(f)) \leq K\}|}$. The notation $|_S$ denotes evaluating f on only the training data in the training set S (as opposed to evaluating on all 2^n datapoints). Plugging this into Eq. (A.16) leads to:

$$\epsilon(Q^*) \leq 1 - \exp\left(-\frac{\ln \frac{|\{f : K_{LZ}(f)) \leq k\}|}{|\{f : K_{LZ}(f)) \leq k, f|_{s=S}\}|} + \ln \frac{2m}{\delta}}{m - 1}\right), \tag{E.25}$$

the implementation of the PAC-Bayes bound for $\mathcal{U}_{\leq K}$.

If at least one function achieves zero training error for some $U_{\leq K_M}$, we can use the realisable PAC Bound adapted from Eq. (A.16) (where we don't make the customary approximate step that $(1-\epsilon)^m \approx e^{-m\epsilon}$ which only holds for small $m\epsilon^2$:

$$\epsilon_G \leq 1 - \exp\left(-\frac{\ln |\mathcal{H}_k| + \ln \frac{1}{\delta}}{m}\right) \sim 1 - \exp\left(-\frac{k \ln 2 + \ln \frac{1}{\delta}}{m}\right). \tag{E.26}$$

where the second relation holds if we assume that $|\mathcal{H}_K| \sim 2^k$. In practice, we take the cumulative measure $|\mathcal{H}_K| = \int_{K_{min}}^{K} P(K')dK'$ using our LZ complexity measure to calculate $P(K)$. This explains why the PAC bound flattens off at larger K

E.2 Further experiments

E.2.1 Experiments using the unbiased learner

For the $n = 5$ systems there are only $2^{32} \sim 4 \times 10^9$ strings so full sampling is relatively straightforward. We (randomly) select a fixed training set S of size m, and sampled over all functions f for several target functions ft, namely

(a) '0011' $\times$ 8 (a simple function);

(b) '01001111000111111110101111110100' (a complex function);

(c) '1001' $\times$ 8 (a lowK lowP function);

(d) the parity function.

The simple functions (a) and (b) are chosen to contrast simple and complex targets. (c) is a simple function which the DNN has difficulty producing; it was not observed in 10^8 random samples. In other words, even though it has low complexity, the DNN is not biased towards it. Such lowK lowP functions are also seen in other input-output maps [66]. A bound can be derived that relates the distance that $\log P(f)$ is below the AIT-inspired upper bound from [65] to the randomness deficit in the set of inputs. In other words, lowK lowP functions have inputs that are simpler than the average inputs. One consequence of this connection is that the total probability of such lowK lowP functions is small, because the vast majority of inputs are complex, and so only a relatively small fraction are available to create such lowK lowP outputs (Note that the arguments in [66] hold for discrete inputs, and here the weights are continuous; nevertheless the arguments can be extended if an arbitrary discretization is applied to the weights). Finally, the parity function is chosen because it is thought that DNNs have a hard time learning it, and so this provides an example of a function that the DNN may be biased against.

For the $n = 7$ system, exhaustive enumeration is harder because the total number of possible functions is so large. Therefore we take a relatively large training set $m = 100$, so that we can exhaustively enumerate over all 2^{28} strings that exactly

fit the training data. When $\mathcal{H}_{\leq K_M}$ is not sufficiently expressive to reach zero training error, we sample 10^6 further strings by perturbing a uniformly sampled integer k in range $0 < k < 10$) labels using the a) true function b) all 0's or all 1's function and c) a randomly generated function with high symmetry, as bases for the perturbation. This generates a distribution over low-complexity functions with low train error. This process produced results that looked qualitatively similar to those seen for the $n = 5$ system (which is exact).

Experiments on these unbiased learners are shown in Figs. 4.1i, E.1 and E.2. They illustrate some aspects of the bias-variance tradeoff for classification of the different possible target functions. Some basic patterns are discussed below.

Firstly, we note that these plots do not show double-descent behaviour as in [196] because we are changing the inductive bias of the model by increasing K_M. We are not fixing the model and changing the ratio of parameters to data.

For K_M much less than the complexity of the target $K_t = K_{LZ}(f_t)$, it will normally be the case that the learner cannot achieve zero training error, and most of the error will be due to bias. As K_M is increased, there will be a $K' \leq K_t$ for which a function in $\mathcal{H}_{\leq K'}$ will first fit S with zero error. For large training sets, K' will roughly equal $K_{LZ}(f_t)$. For very small $|S|$, K' can be quite small, for example, if a function is highly class-imbalanced towards 0's, then for smaller training sets the probability to find an i.i.d. training set with all $0's$ which could be fit by a trivial function $f = 0000.....$ with minimal complexity may not be negligible.

The smallest $\mathcal{H}_{\leq K'}$ with $\epsilon_S = 0$ will have the smallest PAC-Bound on generalization error. In this case, the error due to bias may be small if the main prediction is close to the true complexity; this typically occurs when the true complexity of the function is close to this K', and the error due to variance will also be small (as there are few functions in the posterior).

For $K = K_{max}$, $|\mathcal{H}_{\leq K_{max}}| = 2^{2^n}$. All possible functions on the test set are present and equiprobable, so the mean generalization error will be exactly 0.5. The error due to variance here will be the largest, as there is a huge range of functions

to choose from; depending on the target function the error due to bias can also increase as the learner is no longer biased towards low-complexity functions.

E.2.2 Neural networks with a maximum complexity cut-off K_M

If we restrict hypotheses to $\mathcal{H}_{\leq K}$, then the posterior for DNNs trained on a training set S can be defined as:

$$P_{\leq K_M}(f \mid S) = \frac{P_{SGD}(f \mid S\mathbb{1}(f \in \mathcal{H}_{\leq K})}{\sum_{f'} P_{SGD}(f' \mid S\mathbb{1}(f' \in \mathcal{H}_{\leq K})}.$$

The posterior $P_{\leq K_M}(f \mid S)$ is approximated by training 5000 DNNs to 100% training accuracy (with AdvSGD, batch size 16, and networks of width 40), and rejecting all functions not found in each $\mathcal{H}_{\leq K}$, as with the unbiased learner. For the $n = 7$ system, we used a training set of size 100 when restricting the complexity. For each function that the DNN converges to, that is within the restricted hypothesis class, we calculate the complexity, as well as $\epsilon_G(f)$ and $\epsilon_S(f)$. We record their averages and standard deviations in the plots Figs. 4.1i, E.1 and E.2.

The ReLU-based DNN and the tanh based DNN with $\sigma_w = 1$ have a strong inductive bias towards simple functions, meaning in general the probability to obtain a highly complex f is much smaller than the probability to obtain a simple f. The strong suppression of these complex functions means that removing them with $\mathcal{H}_{\leq K}$ typically has a relatively small impact on generalization error. By contrast, the chaotic neural networks do not suppress these complex functions strongly enough to counteract the exponential growth in $\mathcal{H}_{\leq K}$ with complexity, and thus errors due to variance can be as large as for the unbiased learner.

E.2.3 PAC-Bayes estimates for $n = 5, 7$

The PAC and PAC-Bayes bounds (see Eq. (E.25)) can both be enumerated exactly (as $P_{K_{LZ}}(f) \mid S$ can be calculated exactly for all f) for the $n = 5$ system, and for the $n = 7$ system we use the approximation in Fig. E.3 to calculate $P(K)$, and we can exactly enumerate the numerator in Eq. (E.25) by sampling. The PAC-Bayes

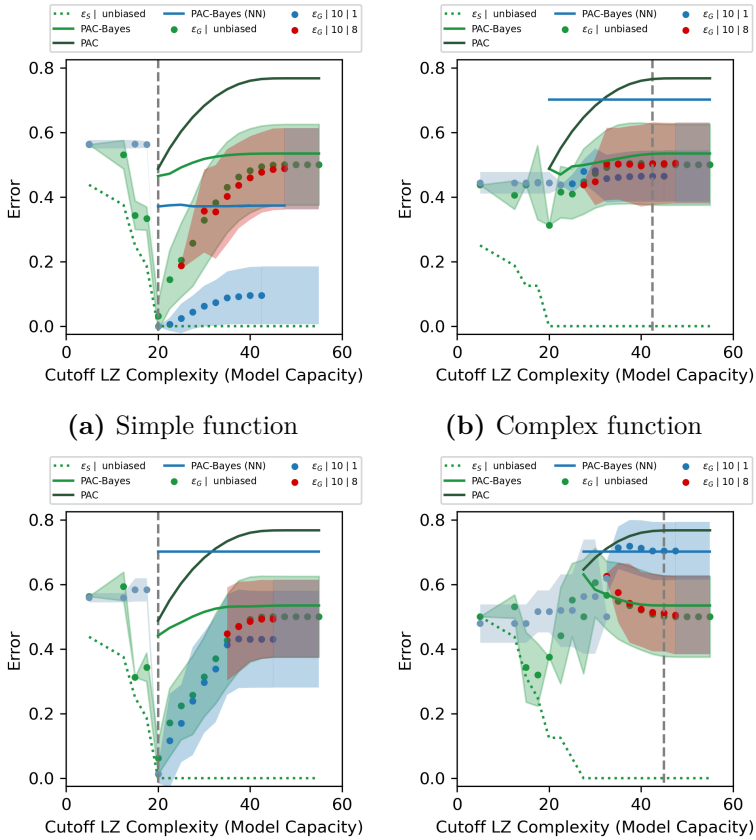
bounds for the neural networks (for the n=5 system) use 10^8 samples to approximate $P(f)$, and $P(S)$ is extracted from this distribution. In the event that no function found in 10^8 samples fits the training set, the bound is plotted with $P(S) = 10^{-8}$, to show the minimum value of the PAC-Bayes bound.

E.3 Selected further experiments

For a complete set of further experiments, see the Supplementary Information in Mingard et al. [5].

E.3.1 Extended data for K-learner with PAC and PAC-Bayes bounds

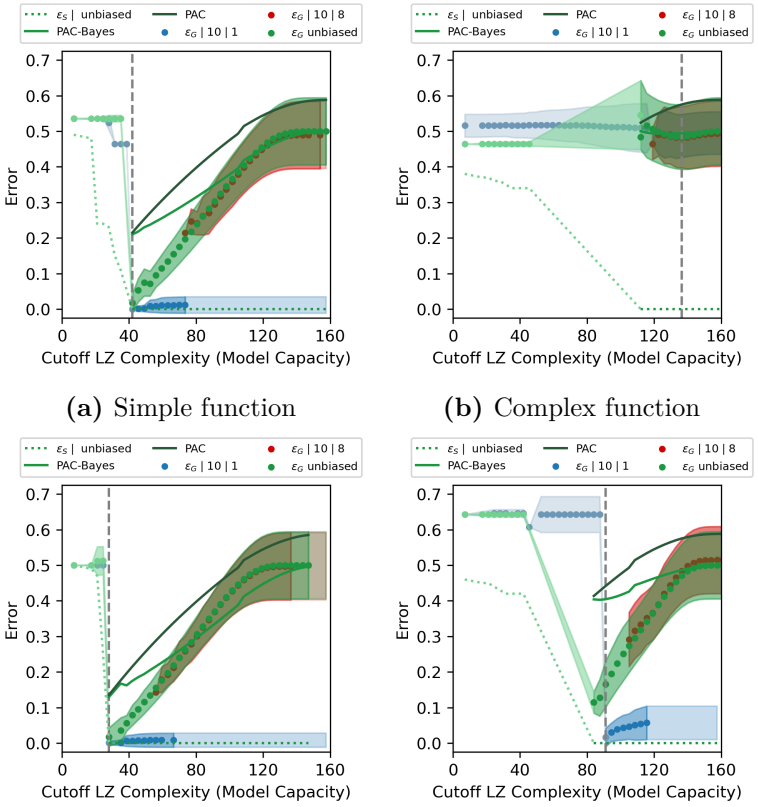
In this section, we expand on Fig. 4.1 (i) for different target functions, as well as for the smaller $n = 5$ system. As can be seen in Fig. E.1 for $n = 5$ and in Fig. E.2 for $n = 7$, there are clear differences in performance on simple rather than complex functions. We also include a lowK lowP function that is simple, but has low probability (and so is far from the bound) (see [66] for a longer discussion of such functions). We would expect (and indeed observe for $n = 5$) that the DNN struggles more with functions for which it has a smaller $P(f)$, and therefore a smaller inductive bias at initialization. Finally, we study the parity function, for which $f = (1, 0)$ if the number of ones in the input is even (odd). It, therefore, has the highest input sensitivity of any function, because any change of input changes the function output. There are well-known questions in the field about whether or not a DNN can learn the parity function [250, 71].



(a) Simple function **(b)** Complex function

(c) low-K low-P **(d)** Parity function

Figure E.1: K-learner for n = 5 Boolean system Training set size $m = 16$. Refer to Appendix E.3.1 for full details on these experiments. $\epsilon_T \mid$, unbiased denotes the minimum training error achievable with functions f of $C_{LZ}(f) \leq K$. $\epsilon_G \mid$, unbiased denotes the test error of those functions, assuming each is equally likely. $\epsilon_G, 10 \mid \sigma_w$ denotes the generalization error of a DNN with 10 layers and weight initialization σ_w, upon training to 100% training accuracy. The lighter blue datapoints are for $\epsilon_G, 10 \mid 1$, with function predictions sampled from the DNN each time training accuracy decreased. The chaotic DNN initializes at high K functions, so low K functions are not reached as part of training (in 10^4 samples). The PAC-Bayes (NN) bound is for the DNN with $\sigma_w = 1$, uses $\delta = 0.01$, and the marginal likelihood is calculated with 0-1 loss using 10^8 samples from the prior of the $\sigma_w = 1$ DNN (divide the number of functions that fit the training dataset S found during sampling by 10^8). See Appendix A.5 for details about the bound. The PAC-Bayes bound is for the unbiased learner (where the marginal likelihood is calculated exactly with 0-1 loss); and the PAC bound is given in Eq. (E.22) Both use $\delta = 0.01$. (a) '0011' $\times$ 8, (b) '01001111000111111110101111110100', (c) '1001' $\times$ 8, (d) Parity function. A simple function (a) is learnable by the uniform learner with a hypothesis space restricted to the complexity of the target function, or the DNN with $\sigma_w = 1$. A highly complex function (b) should not be learnable no matter the chosen training set and hypothesis restriction. (c) A simple function (with respect to LZ complexity) has very low prior probability (was not found during sampling the prior, so $P(f) < 10^{-8}$) is a low-K low-P function. Therefore, the DNN with $\sigma_w = 1$ does not generalise that well either. (d) The parity function is clearly very hard for the DNN to learn. Interestingly it seems biased against it, as it generalises worse than a random learner. However, the parity function is learnable for the $n = 7$ dataset with $m = 100$. (see Fig. E.2).



(a) Simple function **(b)** Complex function

(c) LowK lowP function at $n = 5$ **(d)** Parity function

Figure E.2: K-learner for n = 7 Boolean system See Fig. E.1 for notation. Training set size $m = 100$. For details of the implementation of the different learners, as well as the realisable PAC and PAC-Bayes bounds (which use $\delta = 0.01$), see Appendix E.3.1.
(a) simple function '10011010' × 16
(b) complex function '00110111101001110100101110010111000111001011101110111010111011110000011000110001111111110011101100111101001010101110111100111000 1'
(c)lowK lowP function at $n = 5$ '1001' × 32
(d) Parity function
Unlike the $n = 5$ case, good generalization is achieved for the lowK lowP function in (c) and parity function in (d). This may be due in part to the relatively large training set ($m = 100$).

E.3.2 Extended figures of priors P(K) and P(f) for DNNs

In this section, we plot in more detail than the main text some results for the priors of DNNs upon random sampling of parameters. In Fig. E.3 we show priors over complexity $P(K)$ for an ordered and chaotic FCN with 10 layers, as well as the $P(K)$ for randomly sampled strings of length 128.

In Fig. E.4 we show similar priors (and also $P(f)$) for smaller systems with $n = 5$ and $n = 4$, where the number of functions are $2^{2^5} = 4,294,967,296$ and

$2^{2^4} = 65,536$ respectively, both of which can be fully sampled. The downside of these very simple systems is that the finite-size issues with the LZ complexity measure become more pronounced since the strings are only length 32 and 16, instead of 128 as in the $n = 7$ system. Nevertheless, it is clear that the same qualitative differences between the two DNNs are observed in the $n = 7$ system.

In Fig. E.5 (a) we compare the priors $P(f)$ for DNNs with the more popular ReLU activations functions to that of the tanh-based priors with different σ_w's. Our "standard" $\sigma_w = 1$ prior is very close to the ReLU prior (this also holds for $P(K)$). Moreover, both priors closely follow the idealised Zipfian curve

$$P(f)_{Zipf} = \frac{1}{\ln(N_f)(\text{rank}(f))^\alpha} \tag{E.27}$$

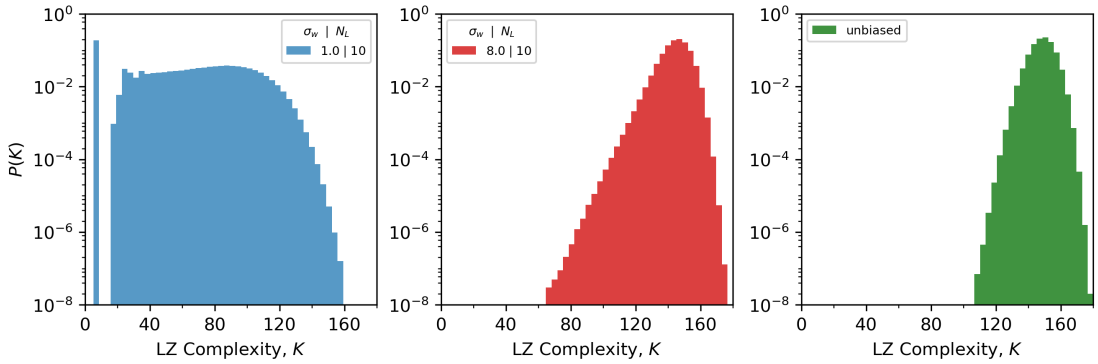
where the number of functions $N_f = 2^{128}$ and with exponent $\alpha = 1$. For larger σ_w, as the system enters the chaotic regime the prior begins to noticeably deviate from the ideal Zipfian curve, which is not surprising given that the fully chaotic fixed-point has a uniform prior over functions as shown in Appendix E.1.12.

To get a sense of the full range probabilities of $P(f)$ vs $K(f)$ we make a simple approximation $P(f) \approx P(K)/|\{f : K_{LZ}(f) = K\}|$, where the numerator comes from the measured $P(K)$ from the first two panels of Fig. E.3(b), and the denominator comes from the third panel in Fig. E.3(b). As shown Fig. E.5(b), this produces an approximate $P(f)$ versus K over a wide range of values. While this approximation is rather crude, it does allow us to get a sense of the scale of $P(f)$ over the full range of K.

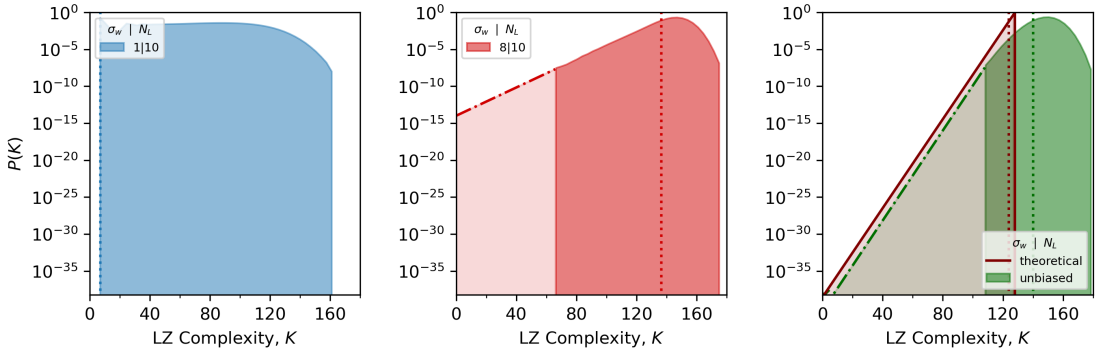
This larger scale helps us see that the chaotic prior with $\sigma_w = 8$ is still quite strongly simplicity biased compared to uniform sampling over functions. Nevertheless, as can be seen in Fig. E.3, this simplicity bias is not nearly enough to counteract the 2^K growth in the number of functions with increasing K. In other words, an Occam's razor-like simplicity bias is not enough for good generalization on simpler functions. It has to be the right strength or else the DNNs will still suffer from a tendency to converge on functions that are too complex, much as one expects from classical learning theory.

In Fig. E.6 we depict the priors for a range of σ_w and depths N_l. For increasing σ_w and increasing N_l, which corresponds to entering more deeply into the chaotic regime, the amount of simplicity bias decreases, and the $P(K)$ plots show a sharper rise towards complex functions.

Finally, in Fig. E.7 we show the priors $P(K)$ for three different complexity measures. The first is the LZ measure used before. The second is the simple binary entropy $K_S = -\log_2 p - \log_2(1-p)$ where p is the fraction of 0's (or equivalently of 1's. Simple strings such as all 0's or all 1s also have low LZ complexity, but strings such as "01010101..." have high binary entropy, but low LZ complexity. The third measure is a classic measure of Boolean complexity, the minimum of the disjunctive/conjunctive normal form. While all three $P(K)$ plots differ in detail, as expected, they also all present the same qualitative behaviour in that the chaotic DNN prior is much closer to that of random strings than the order DNN prior is.

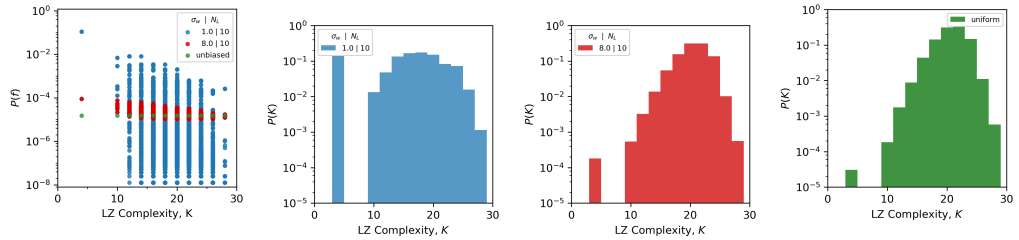


(a) P(K) distributions for LZ complexity (raw data only)



(b) P(K) distributions for LZ complexity extended data

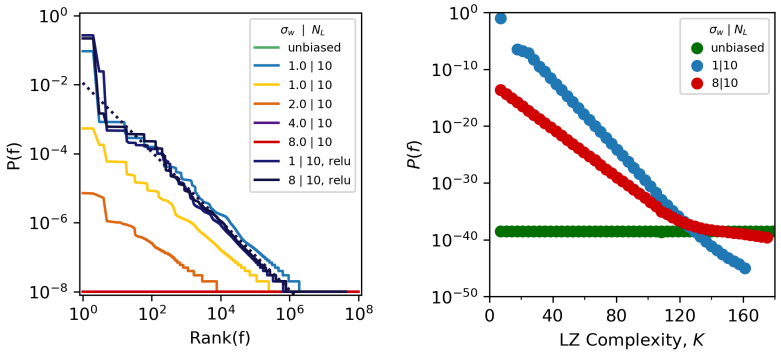
Figure E.3: P(K) vs LZ complexity K for the $n = 7$ Boolean system. Left and centre panes show DNNs with 10 layers, $\sigma_b = 0.2\sigma_w$ and $\sigma_w = 1, 8$ respectively, with tanh activations and hidden layer width 40. The right-hand pane shows $P(K)$ when strings of length 128 are uniformly sampled, for 10^8 random samples. (a) Raw values for LZ complexity (without extrapolation) and (b) with log-linear extrapolation (inspired by plots for other lengths in [65]) here the y-axis is cut off at 2×2^{-128} – the exact value of $P(K)$ for the simplest functions when randomly sampling. The blue, red and green areas show the same data as found in Figs. 4.1g and 4.1h respectively. 90% of the probability mass lies to the right of the vertical dotted lines; the dash-dot lines are lines of best fit for the left-hand two panes; for the right-hand most pane, one point is fixed at $(7, 2 \times 2^{-128})$ (the theoretical value for $P(K = 7) = 2 \times 2^{-128}$, as $K = 7$ corresponds to the strings of all 0s and 1s only), and connected to the end of the sampled area. Max observed LZ complexity is $K = 179$ (in the right-hand plot), x-axis cut at $K = 180$. The red theoretical curve in the right plot is for an ideal compressor for which $P(K) = 2^K$. See Appendix E.1.4 for further experimental details



(a) $P(f)$ vs $K_{LZ}(f)$, n=4 **(b)** n=4, $\sigma_w = 1$ **(c)** n=4, $\sigma_w = 8$ **(d)** n=4, uniform (exact)

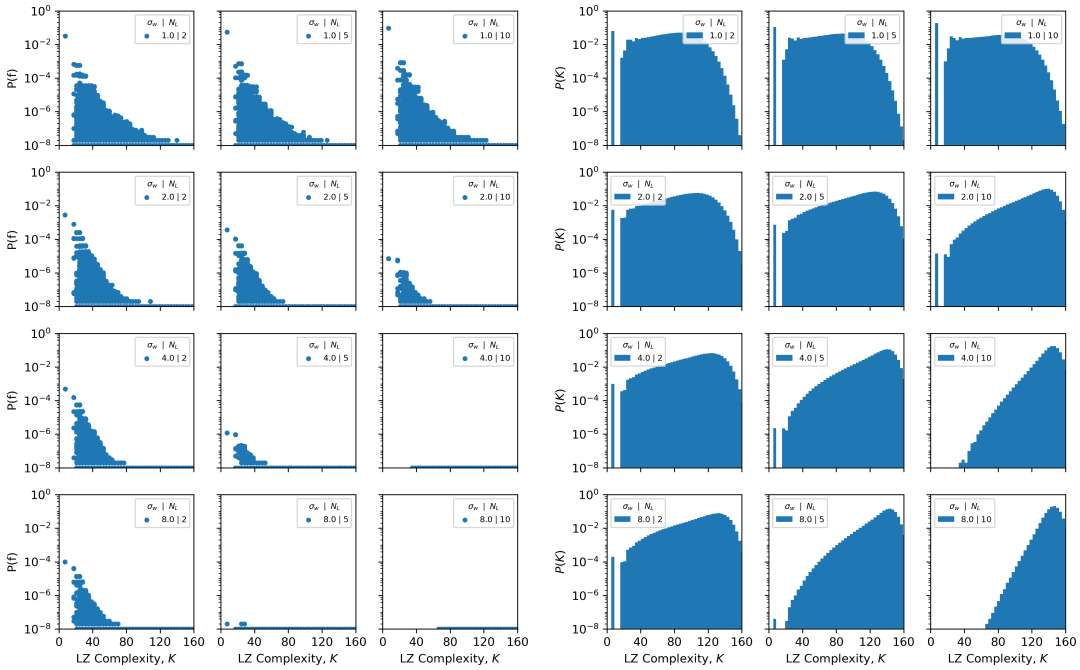
(e) $P(f)$ vs $K_{LZ}(f)$, n=5 **(f)** n=5, $\sigma_w = 1$ **(g)** n=5, $\sigma_w = 8$ **(h)** n=5 uniform (exact)

Figure E.4: Priors upon random sampling of parameters vs LZ complexity K for the n = 5 and n = 4 Boolean systems. Prior $P(f)$ on $\{0,1\}^5$ for 10 layered DNNs with $\sigma_w = 1, 8$ from 10^8 samples. The green points show the theoretical $P(f)$ with a uniform prior over functions. The $n = 4$ system is so small that $P(f)$ for the uniform learner is 2^{-16}, and we can take enough samples to show the functions with $P(f)$ much lower than this for the ordered prior $\sigma_w = 1$, for which generalization would be worse than random, as consistent with no free lunch theorems. The $n = 5$ system is fractionally too large to see these functions, but we are able to get enough samples to see the trivial functions in the $\sigma_w = 8$ prior, and clearly, $P(K)$ grows exponentially in $\sigma_w = 8$, suggesting the approximation in Fig. E.3 is sensible.



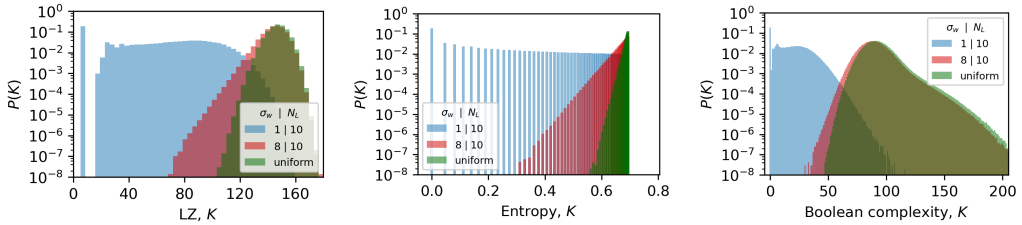
(a) P(f) vs LZ Complexity **(b)** P(f), n=7, (approx)

Figure E.5: Comparing priors over functions P(f) (a) Prior $P(f)$ for 10-layer FCNs with tanh and ReLU activations, on $n = 7$ Boolean functions, ranked by probability of individual functions, generated from 10^8 random samples of parameters $\Theta \sim \mathcal{N}(0, \sigma_w)$. (b) To get a sense of what $P(f)$ vs LZ complexity K looks like on a much larger scale, $P(K)$ data from Fig. E.3 was used to approximate $\langle P(f \mid K) \rangle \approx P(K)/|\{f : C_{LZ}(f) = K\}|$. Note that these points lie below many of the samples in Fig. 4.1b; this is to be expected as the calculation returns the average value for $P(f \mid K)$, which is clearly much lower than $\max P(f \mid K)$ (see e.g. Fig. E.4a). Both the chaotic and the normal DNN are strongly simplicity biased. Nevertheless, as shown previously, the lower simplicity bias of the chaotic prior is not strong enough to overcome the growth of the number of functions with increasing complexity, so the prior over complexity is dominated by large-complexity functions for the chaotic regime.



(a) P(f) vs LZ Complexity **(b)** P(K) vs LZ Complexity

Figure E.6: DNN priors for different σ_w and depth N_l: (a) Empirical probability $P(f)$ of individual functions versus their LZ complexity for networks initialized with various σ_w and number of layers N_l, with hidden layer width 40 and tanh activations for 10^8 random samples. Points with a probability of 10^{-8} are not removed (even though sampling errors for these low probability points are large) because in plots ($\sigma_w = 4$, $N_l = 10$) and ($\sigma_w = 8$, $N_l = 10$) only points of this type are found. (b) *A-priori* probability $P(K) = \sum_{f:K_{LZ}(f)=K} P(f)$ versus LZ Complexity based on 10^8 samples of a neural network with N_l hidden layers, hidden layer width 40 and tanh activations, initalized i.i.d with weight standard deviation σ_w. $\sigma_b = 0.2\sigma_w$. Further examples for Fig. 4.1a. See Appendix E.1.4 for further experimental details



(a) LZ Complexity **(b)** Entropy **(c)** Boolean complexity

Figure E.7: P(K) for different complexity measures. Prior $P(K)$ for 10-layer FCNs with tanh activations, on $n = 7$ Boolean functions, ranked by probability of individual functions, generated from 10^8 random samples of parameters $\Theta \sim \mathcal{N}(0, \sigma_w)$. Uniform sampling is included for reference. (a) LZ Complexity (b) Entropy (c) Boolean complexity. (b) the entropy here is that of the string representation of the function, and (c) is the same Boolean complexity measure as in [64] (the minimum of the Disjunctive/Conjunctive Normal Form, calculated with SOPform and POSform from Scipy).

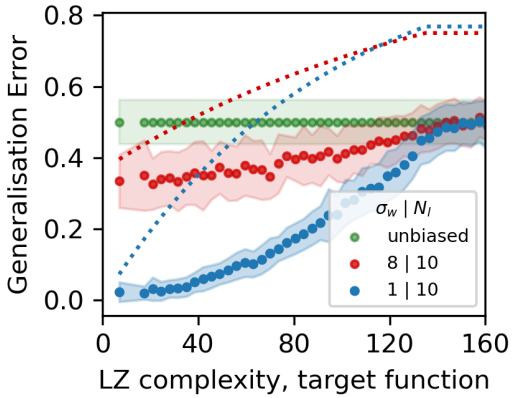


Figure E.8: Fig. 4.1c with PAC-Bayesian bounds We used extrapolated $P(K)$ data from Fig. E.3 to calculate PAC-Bayesian bounds for the task shown in Fig. 4.1c. We assumed that each function in each complexity class has the same probability, meaning the marginal likelihood for a function with complexity K, $P(S)_K \approx P(K)/N_K$, where N_K is the number of functions with complexity K. N_K is estimated from the rightmost plot in Fig. E.3(b), and $P(K)$ from the middle and leftmost plots in the same figure. Note that the PAC-Bayes bound for $\sigma_w = 8$ is lower than that of $\sigma_w = 1$ for the larger complexities – which is to be expected, as careful examination of Fig. E.3 shows a greater prior probability assigned to the more complex functions for $\sigma_w = 8$ than $\sigma_w = 1$.

F

Appendices for Chapter 5

F.1 Datasets, Architectures and Algorithms

F.1.1 Datasets and architectures

Details of the machine learning architectures used for numerical experiments are listed in Table F.1. Unless otherwise specified, all neural network layers are randomly initialized using their respective default methods in PyTorch 2.0.1 [251]. All activation functions are ReLU ($f(x) = \max(0, x)$), unless otherwise specified. Datasets used for numerical experiments are described in Table F.2.

F.1.2 Generating the initialisation prior

To generate the initialisation prior $P(f)$, we use the following procedure. As in the main text, we consider a c-way classification task on a finite set of inputs $S_X = \{x_i\}_{i=1}^m$. The datasets come with true labels $S_Y = \{y_i\}_{i=1}^m$, and $S = \{x_i, y_i\}_{i=1}^m$. Of course, the prior is independent of S_Y. A function f is a labeling on S_X, i.e., $f = \{f_i\}_{i=1}^m$. Both f_i and y_i take values from among the c classes.

We need to approximate the initialisation prior over functions f for a neural network F,

$$P(f) = \int \mathrm{d}\theta \; P_{\mathrm{par}}(\theta) \prod_i (\mathbb{1}[F(X_i;\theta) = f_i]) = \int \mathrm{d}\theta \; P_{\mathrm{par}}(\theta) \mathbb{1}[F(S_X;\theta) = f], \quad \text{(F.1)}$$

Architecture	Details
FCN	(Fully Connected Network/Multi-Layer Perceptron) 3 hidden layers, 128 neurons in each layer (unless otherwise specified)
CNN	(Convolutional Neural Network) 2 convolutional filter layers, receptive field 3×3 pixels, max pool after each convolutional layer, followed by two linear layers
LSTM [33]	(Long-Short Term Memory network) Embedding layer, bidirectional LSTM with 256 neurons, MaxPool1D, linear layer with 512 neurons, linear classifier
Transformer [252]	RoBerTA, a transformer-based neural network followed by a classification layer. In the main text, we use a pretrained RoBerTA and only randomly initialize the last layer. In the appendix (Fig. F.4), the entire network is randomly initialized.
Resnet18 [118]	PyTorch default for CIFAR-10, batch norm layers removed
RBF Kernel GP	Gaussian Process [253] with RBF (Radial Basis Function) kernel, with all hyperparameters set to `scikit-learn` defaults
Perceptron	Linear model, bias initialisation variance is 0.1 of the weight initialisation variance
GCN [254]	(Graph Convolutional Neural Network) 2-hidden layer Graph Convolutional Neural Network

Table F.1: Details of neural network architectures used in numerical experiments.

Dataset	Details
Boolean dataset	The n-dimensional boolean dataset consists of all 2^n binary vectors of dimension n, $\{0,1\}^n$.
Random dataset	Input dimension 256, each element sampled i.i.d. from $\text{NORMAL}(0,1)$ (once sampled, the dataset is fixed).
MNIST [255]	A standard dataset of $\sim 60,000$ images of grayscale handwritten digits, 28×28 pixels each. For experiments with effective priors, the data is centered and normalized.
Fashion-MNIST [256]	A standard dataset of images sorted in 10 fashion-related categories; dataset structure otherwise mirrors MNIST.
CIFAR-10 [154]	A standard dataset of 60,000 colored images sorted into 10 classes, with standard normalization and centering
IMDb [257]	Dataset of 50,000 movie reviews, classified as positive or negative
CiteSeer [258]	Dataset of $\sim 3,300$ scientific publications classified into 6 topics and represented by presence or absence of one of $\sim 3,700$ unique words (dataset taken from Pytorch Geometric).

Table F.2: Details of datasets used in numerical experiments.

where $P_{\text{par}}(\theta)$ is the parameter initialisation distribution of F. $F(S_X;\theta)$ is the post-thresholded (for binary classification) or argmaxed (for multiclass) output of the network, i.e., an integer denoting the class. $P(f)$ is the probability that the DNN expresses f on S_X upon initialisation (before an optimization process). Typically, the total number of samples taken is 10^8.

Here is Pythonic pseudo-code for generating the prior $P(f)$:

```
# Bayesian prior at initialization
from collections import Counter
import Data # input data
import DNN # deep neural network

def sample_function(network: DNN, Sx: Data):
    network.initialize_parameters()
    f = network.predict(Sx) # get network prediction on data Sx at init
    return f

Sx = Data()
network = DNN()
f_frequency = []
for count in range(total_samples):
    f = sample_function(network, Sx)
    f_frequency.append(f)

Prior_f = dict(Counter(f_frequency)) # dictionary of frequency of functions
Prior_f = {i:j/total_samples for i, j in Prior_f.items()} # dictionary of P(f)
```

F.1.3 Fitting normalization of the rank-ordered plots

As discussed in the main text, we wish to compare our data to *normalized* power-law distributions.

There are 2^m possible unique functions when the task is binary classification. If $P(f)$ were exactly Zipfian, we could approximate the normalization constant

$$1 \approx A \int_1^{2^m} \frac{1}{x}\, \mathrm{d}x\,, \tag{F.2}$$

which gives $A = 1/(m \ln 2)$. Here $\approx$ denotes approximation by replacing the sum over discrete functions by an integral.

We typically find, however, that the low-rank functions are very non-Zipfian. Such deviations at low ranks are common when power laws are fit to empirical data [259]. The arguments for the emergence of Zipf's law in the thermodynamic limit with latent fields similarly say little about the non-asymptotic behavior at

low ranks [108]. These low-rank effects make it difficult to fit empirical data to power laws, and specifically to evaluate the normalization constant.

We can address this issue when performing fits by noticing that, if Zipf's law is observed at high ranks, $r > s$, then:

$$1 - \sum_{r=1}^{r=s} P(r) \approx A(s) \int_s^{2^m} \frac{1}{x}\,\mathrm{d}x\,, \tag{F.3}$$

giving

$$A(s) \approx \frac{1 - \sum_{r=1}^{r=s} P(r)}{m \ln 2 - \ln s}. \tag{F.4}$$

In most fits, only the functions with rank $f = 1, 2$ are outliers, and it is sufficient to take $s = 2$ to evaluate A. As expected if Zipf's law holds for $r > s$, we find that fits are generally insensitive to this choice of s (Fig. F.5).

An analogous calculation can be made for multi-class classification problems, giving

$$A(s) \approx \frac{1 - \sum_{r=1}^{r=s} P(r)}{m \ln c - \ln s}. \tag{F.5}$$

F.2 Effective priors for SGD & Loss-based likelihoods

In Appendix F.5 and Section 5.3, we prove that the prior $P(f)$ of a Bayesian learning agent over a training set must satisfy Zipf's law for optimal learning in the large-data limit. The proof relies on the likelihood taking the form

$$P(S|f) \propto \exp\left(-\kappa m_{\text{train}} \epsilon_{\text{train}}\right) \tag{F.6}$$

for some constant $\kappa \geq 0$, where ϵ_{train} is the fraction of errors made on the training set S of size m_{train}. The 0–1 likelihood is a particular limit of this class, obtained by taking $\kappa \to \infty$. We will refer to likelihoods that only depend on the training error as *error-based* likelihoods (note that we anticipate that it should be possible to extend our proofs to a broader class of likelihoods). We show in Fig. 5.1 that the initialization prior $P(f)$ of a wide range of DNNs and datasets satisfies Zipf's law. Therefore, if those neural networks were trained using Bayesian inference with $P(f)$

as a prior and Eq. (F.6) as the likelihood, they would thus satisfy the necessary conditions for good learning shown in the main text.

However, there are two layers of approximations we need to explore to relate this to neural networks fully

1. Kernel approximations to neural networks rely on loss-based likelihoods, which use the *continuous* output of the network, even when they are trained using Bayesian inference.

2. Neural networks are usually trained using a method like stochastic gradient descent, which uses a continuous loss (as in 1), and further is *not* Bayesian.

So, if we want to apply the initialization prior $P(f)$ and our proof to real neural networks, we would need to relate the effect of SGD with Bayesian inference to our highly idealized training algorithm.

To do this, we will first explore 1) – how does using a continuous likelihood affect the posterior? To do this, we will introduce the idea of the *effective prior*: the prior that an error-based Bayesian model would need to have in order to replicate the true posterior generated by a loss-based model. Having understood how the effective prior behaves on the Bayesian kernel method, we then turn to real neural networks.

The main takeaways are as follows

1. We note that neural networks become kernel methods (which for our purposes are interchangeable with linear models) in the appropriate infinite width limit—where training with the mean-squared-error loss becomes equivalent to sampling from a prior with a continuous (Gaussian) likelihood.

2. With finite data, a continuous likelihood can "collapse" the posterior: unlike error-based likelihoods, which always preserve some posterior variance, loss-based likelihoods can be made arbitrarily sharp, eliminating uncertainty and collapsing the posterior onto a single function (the MAP solution). *However*, this behaviour arises only under the unrealistic assumption of essentially noise-free data and, in our experiments, does not improve generalisation.

3. Furthermore, in the limit of infinite data, however, this "collapse" may not persist. In particular, we show that a given loss-based likelihood does not produce zero training error for large m_{train}, suggesting that loss-based likelihoods will not actually produce narrower posteriors than the 0–1 likelihood for large training sets.

4. We then introduced the effective prior: the prior that an error-based Bayesian model would need to have in order to replicate the true posterior generated by a loss-based model. We discovered that in cases where the loss-based likelihood is not too strong—that is, up to the point it achieves 0 training error—the effective prior remains Zipfian. However, once the likelihood becomes strong enough to collapse the posterior beyond 0 training error, the effective prior becomes heavily distorted and "super-Zipfian." This distortion is an artifact, showing that an error-based framework struggles to capture the certainty of a collapsed posterior.

5. SGD exhibits similar behaviour: We extended this analysis to a practical setting by studying a neural network trained with SGD. We found that increasing the learning rate—which is analogous to a stronger likelihood penalty—makes the effective prior increasingly super-Zipfian, just like increasing the strength of the loss-based likelihood in our linear models.

F.2.1 Linear Models/Kernel Methods as a Case Study

Example setup

We consider an overparameterised linear model, with number of parameters $p \gg m_{\text{train}}$ for a binary classification task. The raw output of this linear model is $g(x;\theta)$ and the thresholded output $f(x;\theta) = \mathbb{1}[g(x;\theta) > 0]$. The dataset is $S = \{(x_1, y_1), \ldots, (x_{m_{\text{train}}}, y_{m_{\text{train}}})\}$, with labels $y_j \in \{-1, +1\}$.

The model consists of (1) a fixed feature map $\varphi : \mathcal{X} \to \mathbb{R}^p$ and (2) a learnable weight vector $\theta \in \mathbb{R}^p$. The model's continuous output is:

$$g(x;\theta) = \varphi(x) \cdot \theta \tag{F.7}$$

The final classifier is determined by the sign of this output: $f_\theta(x) = \text{sign}(g(x;\theta))$.

While this is a classification problem, we follow the common practice of training by minimizing a regression loss—the mean squared error (MSE) between $g(x_j;\theta)$ and the integer labels y_j, combined with L2 regularization (ridge regression):

$$L(S;\theta) = \frac{1}{2\sigma^2}\|\theta\|^2 + \frac{1}{2\zeta^2}\sum_{j=1}^{m_{\text{train}}}(y_j - \varphi(x_j)\cdot\theta)^2. \tag{F.8}$$

The minimum of this loss is equivalent to finding the posterior mean of

$$P_{\zeta,\sigma}(\theta|S) \propto \underbrace{\exp\left(-\frac{1}{2\sigma^2}\|\theta\|^2\right)}_{\text{Gaussian Prior } P(\theta)} \underbrace{\exp\left(-\frac{1}{2\zeta^2}\sum_{j=1}^{m_{\text{train}}}(y_j - \varphi(x_j)\cdot\theta)^2\right)}_{\text{Gaussian Likelihood } P(S|\theta)}. \tag{F.9}$$

This probabilistic framing is natural for many optimisers. For instance, we find empirically that training with SGD approximates sampling from this posterior in wide neural networks [2]. The hyperparameters σ and ζ control the variance of the prior and the likelihood, respectively.

From this parameter posterior, we can define a prior and posterior over the discrete functions f:

$$P(f) = \int d\theta P(\theta)\mathbb{1}\left[f_\theta = f\right], \tag{F.10}$$

$$P_{\zeta,\sigma}(f|S) = \int d\theta\, \mathbb{1}\left[f_\theta = f\right] P(\theta|S). \tag{F.11}$$

Key Point 1: The prior over functions, $P(f)$ is independent of σ. **Key Point 2:** Unlike the error-based case, this posterior $P_{\zeta,\sigma}(f|S)$ depends on the continuous values of $g(x;\theta)$, not just the classification errors. (If we write $P(\theta|S) = \exp\left(-\kappa\sum_i \mathbb{1}\left[f(x_i;\theta) = y_i\right]\right) \equiv \exp\left(-\kappa m_{\text{train}}\epsilon\right)$, we see that θ can be integrated out.) The dependence on θ cannot be integrated out to leave a simple function of the error count. The question now becomes:

Can we find a correspondence between the error-based hyperparameter κ and the loss-based hyperparameters ζ and σ?

For a linear model with a Gaussian prior, the posterior predictive distribution for the continuous output $g(x_*)$ on a test point x_* is also a Gaussian, $g(x_*)|S \sim$

$\mathcal{N}(\mu(x_*), \sigma^2(x_*))$, with mean and variance:

$$\mu(x_*) = k(x_*, S_X)^\top (K + (\zeta/\sigma)^2 I)^{-1} Y, \tag{F.12}$$

$$\sigma^2(x_*) = \sigma^2 \left(k(x_*, x_*) - k(x_*, S_X)\right)^\top (K + (\zeta/\sigma)^2 I)^{-1} k(S_X, x_*). \tag{F.13}$$

Here, Y is the vector of labels, $k(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$ is the kernel function, and K is the $m_{\text{train}} \times m_{\text{train}}$ Gram matrix on the training data, with $K_{ij} = k(x_i, x_j)$. There are several limiting cases that we will find interesting in the empirical section:

1. $\sigma \to 0, \zeta \to 0$ **(fixed $\zeta/\sigma = c$):** The posterior predictive mean $\mu(x_*)$ remains fixed, but the variance $\sigma^2(x_*) \to 0$. The posterior distribution over functions collapses to a single function: the MAP solution.

2. $\zeta \to \infty$ **(fixed σ):** The likelihood term in Eq. (F.9) flattens to a constant. The posterior predictive mean $\mu(x_*) \to 0$ and the variance $\sigma^2(x_*) \to \sigma^2 k(x_*, x_*)$, which is simply the prior variance. This removes the influence of the data and is **exactly equivalent to the error-based likelihood with $\kappa = 0$.**

3. $\zeta \to 0$ **(fixed σ):** The likelihood term in Eq. (F.9) sharpens to a delta function. The posterior predictive mean $\mu(x_*) \to k(x_*, S_X)^\top K^{-1} Y$ and the variance $\sigma^2(x_*)$ remains finite. This removes the regularizing effect of σ, but we still have some variance as long as ζ is finite.

Why are these limits interesting? Limit 1 fixes the posterior mean and shrinks the posterior variance by simultaneously shrinking the prior variance and the likelihood relative to the fixed target scale Y. Key point 1 tells us that we can change σ arbitrarily without affecting $P(f)$ (so the posterior of an error-based likelihood is independent from σ in any case). This means we can fix the prior $P(f)$ but shrink the variance in the posterior as much as we want—thus collapsing the posterior to a single classification f. This is not possible with error-based likelihoods (Key point 2). **So the answer to our question is: no, not in general.**

Can we equate the error-based likelihood $P_\kappa(f|S)$ to a loss-based likelihood $P_{\sigma,\zeta}(f|S)$ in any case? Yes, we can (Limit 2). What about choices of σ, ζ in between

these two limits? While we have a closed-form loss-based likelihood posterior, given by $\mu(x_*)$ and $\sigma^2(x_*)$, we do not have an equivalent closed-form solution for error-based likelihoods. This means there is no exact formula for relating some choice of ζ, σ to some κ.

Link between λ and ϵ_S as $m_{\text{train}} \to \infty$

For any fixed m_{train}, taking $\lambda \to 0$ results in zero training error, and a posterior which is even more "concentrated" than that produced by the 0–1 likelihood. However, as we show below, for any particular λ, a large enough training set size produces nonzero training error.

Consider a training set $g_S \equiv (g(x_1;\theta), \ldots, g(x_{m_{\text{train}}};\theta))^\top$ with labels $Y \equiv (y_1, \ldots, y_{m_{\text{train}}})^\top$. Under the Gaussian prior/likelihood in Eq. (F.9), the posterior at the training points is

$$g_S \mid S \sim \mathcal{N}(\mu, \Sigma), \qquad \mu \equiv K\Big(K + (\zeta/\sigma)^2 I\Big)^{-1} Y, \qquad \Sigma \equiv \Big((\sigma^2 K)^{-1} + \zeta^{-2} I\Big)^{-1} \preceq \zeta^2 I. \tag{F.14}$$

(By $A \preceq B$, we mean that $B - A$ is positive semidefinite.)

Define the residual r and (training) margin γ

$$r \equiv Y - \mu = (\zeta/\sigma)^2\Big(K + (\zeta/\sigma)^2 I\Big)^{-1} Y, \tag{F.15}$$

$$\gamma \equiv \min_j y_j \mu_j = \min_j y_j(y_j - r_j) \geq 1 - \|r\|_\infty, \tag{F.16}$$

where we use the fact that $y_j \in \{-1, 1\}$ in the derivation of Eq. (F.15). We can bound r above with

$$\|r\|_\infty \;\leq\; \frac{(\zeta/\sigma)^2}{(\zeta/\sigma)^2 + \lambda_{\min}(K)} \sqrt{m_{\text{train}}}.$$

For any $\delta \in (0, 1)$, a union bound with Gaussian tails gives a bound on the probability that at least one output will have the wrong sign:

$$\begin{aligned}
\Pr\Big(\exists j :\; y_j g(x_j;\theta) \leq 0\Big) &\leq \sum_{j=1}^{m_{\text{train}}} \Pr\Big(y_j g(x_j;\theta) \leq 0\Big) \leq \sum_{j=1}^{m_{\text{train}}} \exp\left(-\frac{\gamma^2}{2\Sigma_{jj}}\right) \\
&\leq \sum_{j=1}^{m_{\text{train}}} \exp\left(-\frac{\gamma^2}{2\zeta^2}\right) \quad \text{since } \Sigma_{jj} \leq \zeta^2 \text{ (by Eq. (F.14))} \\
&= m_{\text{train}} \exp\left(-\frac{\gamma^2}{2\zeta^2}\right).
\end{aligned}$$

Fix $\varepsilon \in (0,1)$ and require $\gamma \geq 1-\varepsilon$. To obtain zero training error with probability at least $1-\delta$, the following is sufficient

$$\frac{\zeta^2}{\sigma^2} \leq \frac{\varepsilon}{\sqrt{m_{\text{train}}}-\varepsilon}\lambda_{\min}(K) \quad \text{and} \quad \zeta \leq \frac{1-\varepsilon}{\sqrt{2\log(m_{\text{train}}/\delta)}} \quad . \tag{F.17}$$

We derive the first inequality by making sure the residual $\|r\|_\infty$ is smaller than ϵ. The second inequality ensures that the probability of making at least one error is less than δ. This derivation starts with the probability bound and uses the fact that we've already forced the margin γ to be at least $1-\epsilon$. In which case $\Pr(\epsilon_S = 0) \geq 1-\delta$. From Eq. (F.17),

$$\zeta_{m_{\text{train}}} = o\left(\min\left\{\frac{1}{\sqrt{\log m_{\text{train}}}},\ \sigma\sqrt{\frac{\lambda_{\min}(K_{m_{\text{train}}})}{\sqrt{m_{\text{train}}}}}\right\}\right) \quad \text{as } m_{\text{train}} \to \infty. \tag{F.18}$$

Asymptotics: Error-based vs. Loss-based Likelihoods

The 0–1 likelihood eliminates posterior uncertainty. Assume data are generated noiselessly by a *true* classifier $f^\star$ and let P_X denote the input distribution. Write the 0–1 (Gibbs) posterior over classifiers f as

$$\Pi_n(df) \ \propto\ \Pi_0(df)\,\exp\Big(-\kappa\, n\,\hat{\epsilon}_n(f)\Big), \qquad \hat{\epsilon}_n(f) \equiv \frac{1}{n}\sum_{j=1}^{n}\mathbb{1}\{f(x_j)\neq f^\star(x_j)\},$$

with fixed $\kappa > 0$ and prior Π_0. By the law of large numbers, $\hat{\epsilon}_n(f) \to \epsilon(f) \equiv P_X\Big(f(X)\neq f^\star(X)\Big)$ almost surely. Hence, for any f with $\epsilon(f) > 0$,

$$\frac{d\Pi_n}{d\Pi_0}(f) \ =\ \exp\Big(-\kappa n\hat{\epsilon}_n(f)\Big) \ \xrightarrow[n\to\infty]{}\ 0.$$

If (i) the model is *realizable* (there exists at least one f with $\epsilon(f) = 0$) and (ii) *identifiable* (the set $\{f : \epsilon(f) = 0\}$ equals $\{f^\star\}$ up to a Π_0-null set), then the posterior mass concentrates at $f^\star$:

$$\Pi_n\Big(\{f^\star\}\Big) \ \xrightarrow[n\to\infty]{}\ 1$$

Let $Z_n(x_*) \equiv \mathbb{1}\{f(x_*) = +1\}$ when $f \sim \Pi_n$. Then $Z_n(x_*)$ is Bernoulli with parameter $p_n(x_*) \equiv \Pi_n\Big(f(x_*) = +1\Big) \to \mathbb{1}\{f^\star(x_*) = +1\}$. Therefore,

$$\Sigma^{01}_{**|S_n} \ =\ \mathrm{Var}_{\Pi_n}\Big(Z_n(x_*)\Big) \ =\ p_n(x_*)\Big(1-p_n(x_*)\Big) \ \xrightarrow[n\to\infty]{}\ 0,$$

i.e., posterior predictive uncertainty vanishes pointwise and hence as a matrix on any finite test set.

Gaussian likelihood with fixed $\lambda > 0$ and $\sigma^2 = 1$ retains some variance. Consider the kernel method with $\lambda^2 > 0$ (we set $\sigma^2 = 1$ wlg). For a test point x_*, let $k_* \equiv k(S_X, x_*)$ and $K \equiv k(S_X, S_X)$. The standard predictive decomposition distinguishes the latent output $g(x_*)$ and the noisy observation $y_* = g(x_*) + \varepsilon_*$, $\varepsilon_* \sim \mathcal{N}(0, \lambda^2)$:

$$\underbrace{\mathrm{Var}\big[g(x_*) \mid S_n\big]}_{\sigma^2_{f,n}(x_*)} = k(x_*, x_*) - k_*^\top \big(K + \lambda^2 I\big)^{-1} k_*, \qquad \mathrm{Var}\big[y_* \mid S_n\big] = \sigma^2_{f,n}(x_*) + \lambda^2.$$

The sequence $\sigma^2_{f,n}(x_*)$ is monotone nonincreasing and nonnegative. Under mild richness of the design (e.g., S_X becomes dense in a compact domain and k is strictly positive definite / universal), one has $\sigma^2_{f,n}(x_*) \to 0$ for each fixed x_* (intuitively: the latent function gets pinned down). Consequently, on any finite test set with conditionally independent noise across test points,

$$\lim_{n\to\infty} \Sigma^{\mathrm{G}}_{**|S_n}(\lambda) = \lim_{n\to\infty} \mathrm{Cov}\big[y_* \mid S_n\big] = \lambda^2 I \equiv \mathbf{C} \succ 0.$$

Thus, with a *fixed* $\lambda > 0$ the predictive covariance for the *observed labels* cannot collapse: it is lower-bounded by the noise floor λ^2. (Equivalently: even though the posterior over the latent g can concentrate, the observable y keeps an irreducible variance unless $\lambda = \lambda_n \to 0$.)

Remarks

The 0–1 likelihood represents the strongest form of error-based likelihood, in the sense that it assigns nonzero posterior weight to the smallest possible set of functions—specifically, $2^{m_{\text{test}}}$ functions. In contrast, a loss-based likelihood admits a limit ($\sigma \to 0$) in which the posterior variance is completely eliminated, assigning nonzero probability to only a single function.

At finite m_{train}, this illustrates that a practical learning agent trained via loss minimization can remove the variance-related errors that would otherwise persist for a purely error-based Bayesian learner. By collapsing the posterior onto the MAP solution, the learning problem effectively reduces from Bayesian

inference to optimisation. In this sense, such agents can overcome the variability introduced by a sub-Zipfian prior.

In the infinite-data limit, however, this "collapse" does not persist for fixed λ. Specifically, we show that a given loss-based likelihood fails to achieve zero training error as $m_{\text{train}} \to \infty$, implying that it does not produce narrower posteriors than the 0–1 likelihood for sufficiently large training sets. Moreover, taking $\lambda \to 0$ corresponds to assuming the data are arbitrarily noise-free. If an ML agent could systematically improve by continually reducing variance through decreasing λ, we would expect ensembles of weak classifiers (e.g., with 51/49 accuracy splits) to perform increasingly well as λ decreases. Since this is not observed empirically (and see the next section for further empirical evidence), we suggest that the vanishing-λ limit does not enhance generalisation and is therefore of limited practical interest.

F.2.2 Linear models/Kernel methods: empirical results

Test and train errors as a function of σ and ζ

We now empirically test the theoretical insights from the previous section using the Neural Network Gaussian Process (NNGP) corresponding to a fully-connected network (FCN) on the MNIST dataset. This framework allows us to compute the posterior over functions exactly, providing a clean setting to study how the posterior behaves as we vary the hyperparameters governing the prior and likelihood. Following Eq. (F.9), the prior variance is controlled by σ^2, while the likelihood strength is determined by the observation noise ζ^2. In Fig. F.1, we visualize how these parameters affect train/test accuracy and the entropy of the model's predictive distribution under two different scaling schemes.

First, Fig. F.1(a,b) examines the case where the ratio σ/ζ is held constant (Case 1), which fixes the posterior mean. In this regime, changes in performance arise purely from variations in posterior variance, governed by σ. While both training and test accuracies saturate for $\sigma^2 \lesssim 10^{-1}$, the entropy of the predictive distribution continues to decrease, indicating that the posterior becomes increasingly

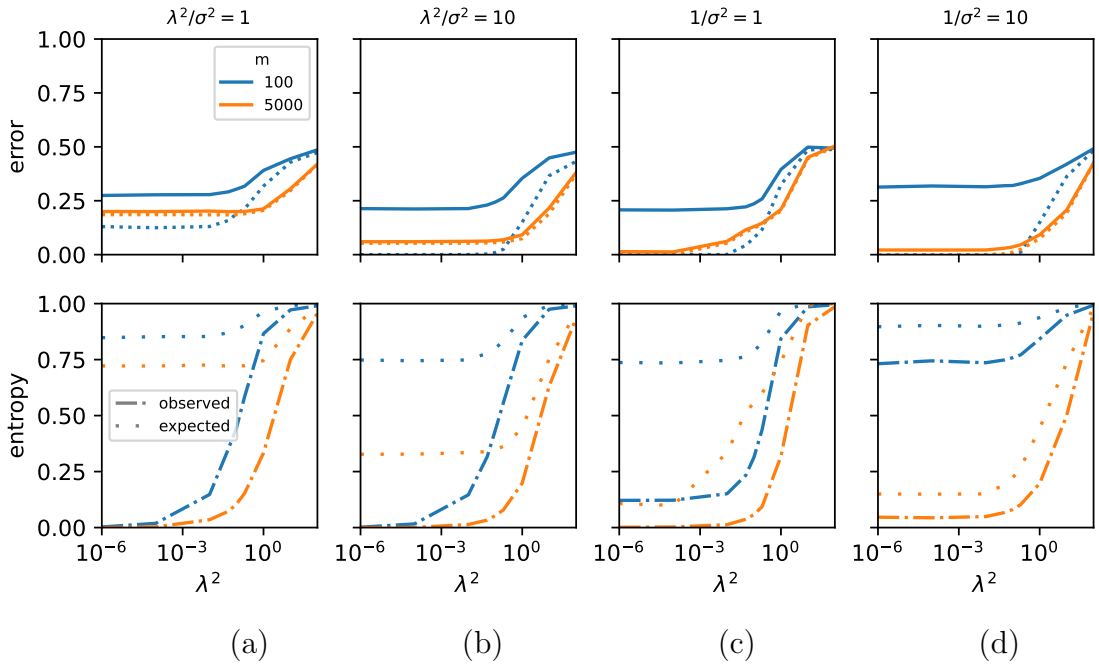
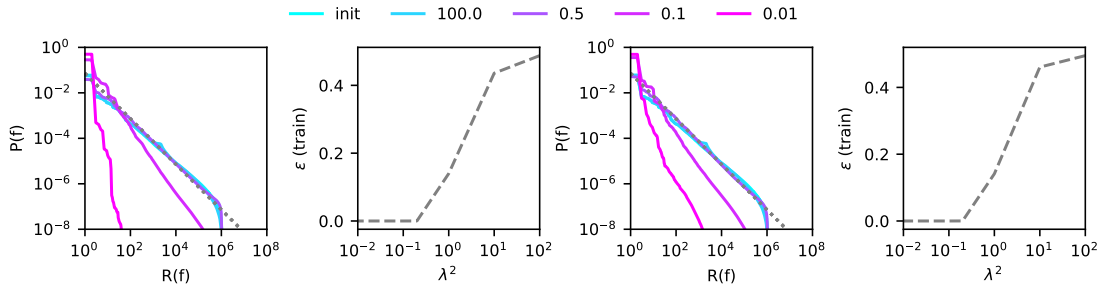


Figure F.1: The effect of ζ on the prior and posterior of an NNGP. We analyze the posterior of an NNGP for a 1-hidden-layer FCN on MNIST. The top row shows train (dotted) and test (solid) errors. The bottom row shows the average entropy of the integer binary classifications (per image). The 'expected' entropy is what a model making random guesses with the same error rate would have; a lower 'observed' entropy signifies higher confidence. This entropy gap signifies the model is highly overconfident. **(a, b)** Here, the ratio ζ/σ is fixed, which keeps the posterior mean constant. Decreasing the prior variance σ^2 scales down the posterior variance, causing the posterior to concentrate around the MAP solution. While accuracy saturates, the prediction entropy continues to drop, indicating that the model's confidence increases even after performance plateaus. **(c, d)** Here, the prior variance σ^2 is fixed while the likelihood's noise term ζ^2 is varied. A smaller ζ^2 corresponds to a stronger likelihood penalty. This alters the posterior mean, pulling it towards a function that better fits the data and improving both train and test accuracy.

concentrated around a single function. This reflects a steady increase in model confidence, even after classification accuracy has plateaued.

In contrast, Fig. F.1(c,d) explores the case where the prior variance (σ^2) is fixed and the likelihood strength (ζ^2) varies independently (Case 2). Here, decreasing ζ alters the posterior mean, moving it further from the prior. Comparing panels (c) and (d) shows that, for a given σ, reducing ζ consistently improves both training and test accuracy, as the model more closely fits the data.

Taken together, these results empirically illustrate the key distinction discussed



(a) $(\zeta/\sigma)^2 = 1$ (b) $(\zeta/\sigma)^2 = 1$ (c) $(1/\sigma)^2 = 1$ (d) $(1/\sigma)^2 = 1$

Figure F.2: The Effective Prior Becomes Super-Zipfian in the Strong Likelihood Limit. Using the same NNGP setup from Fig. F.1, we compute the effective prior $\tilde{P}(f)$. This is the prior that an error-based model (with some likelihood $e^{-\kappa_{\text{eff}} m_{\text{train}} \epsilon}$) would need to have in order to replicate the true posterior generated by the loss-based model. (a) shows $\tilde{P}(f)$ for $(\sigma/\zeta)^2 = 1$, and (b) the corresponding train errors during the creation of the effective prior. (c,d) are the corresponding figures for $(1/\zeta)^2 = 1$. In regimes with high training error $\sigma^2 > 1$, the effective prior is Zipfian, matching the true initialization prior. However, as the likelihood strengthens, it assigns high probability to a much smaller set of functions $g(x;\theta)$ and training error approaches zero (around $\sigma^2 = 0.5$). As σ^2 continues to decrease, functions with high likelihood represent a much smaller part of space concentrated around the train labels than any error-based likelihood (even 01 likelihood). At this point, the effective prior becomes heavily distorted and super-Zipfian, compensating for the approximation's shortcomings.

earlier between continuous (e.g., MSE) and discrete, error-based (0–1) likelihoods. Continuous losses allow the posterior to concentrate on an arbitrarily small set of functions—ultimately collapsing to a single MAP solution—whereas error-based likelihoods treat all zero-error functions equally, preserving posterior variance. *However*, after achieving zero training error (as possible under the 0–1 likelihood), further collapse of the posterior yields no significant improvement in generalisation. This supports our theoretical claim that the vanishing-variance regime confers no practical advantage beyond perfect training accuracy.

Effective priors

To bridge the gap between continuous (loss-based) and discrete (error-based) likelihoods, we seek an **effective error-based model** that best approximates the true posterior induced by the loss-based likelihood. This approximation involves two key components:

1. An **effective likelihood**, parameterized by a single scalar κ_{eff}, which captures the overall penalty applied to training errors.

2. An **effective prior**, $\tilde{P}(f)$, defined as the prior that, when combined with this effective likelihood, reproduces the posterior distribution over functions obtained from the true loss-based likelihood.

We develop approximate procedures for constructing such effective priors (Algorithms 10 and 11). The simplest cases arise when the training error is close to 0.5 or close to 0, which we analyze in detail. In the limit where the loss-based likelihood provides no information (i.e., $\zeta \to \infty$), the match between the two models is exact: we can set $\kappa_{\text{eff}} = 0$, and the effective prior coincides with the original prior, $\tilde{P}(f) = P(f)$.

The situation changes dramatically when the likelihood becomes highly concentrated. As shown in Fig. F.1, the posterior eventually collapses onto a single function (the posterior mean), whereas the strongest possible error-based likelihood corresponds to $\kappa_{\text{eff}} \to \infty$ (the 0–1 likelihood). We compute and visualize these effective priors in Fig. F.2. Panels (a) and (c) depict the estimated effective priors for the two hyperparameter scaling schemes, with corresponding training errors shown in (b) and (d).

In the weak-likelihood regime (large σ^2 or ζ^2), where the training error remains high, the effective prior closely resembles the true Zipfian initialization prior—consistent with an effective likelihood characterized by $\kappa_{\text{eff}} = 0$. However, as the likelihood strength increases and the training error approaches zero, discrepancies emerge. By $\zeta^2 = 0.1$, the loss-based posterior becomes substantially more concentrated than the 0–1 likelihood can capture. To compensate, the effective prior becomes super-Zipfian and deviates from a pure power law. As expected, we see the limits of the error-based approximation: when forced to reproduce a collapsed posterior, the prior must absorb all of the excess concentration that the 0–1 likelihood cannot express.

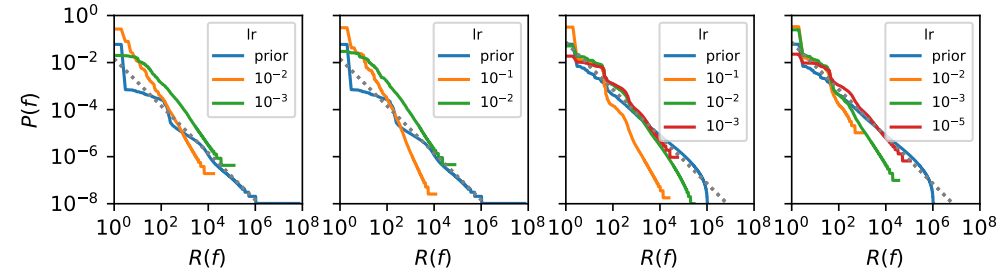
Finally, we note that in the extreme regime where the loss-based likelihood becomes excessively sharp (very small λ), the previous section shows that the posterior collapse does not yield better generalisation. This regime effectively assumes noise-free data, an assumption that is rarely justified in realistic learning scenarios. Hence, while the effective prior must become super Zipfian to reproduce an increasingly concentrated posterior, this behaviour reflects overconfidence rather than improved predictive ability. In practice, driving the loss-based likelihood toward zero noise merely eliminates epistemic uncertainty without enhancing model generalisation.

F.2.3 SGD prior

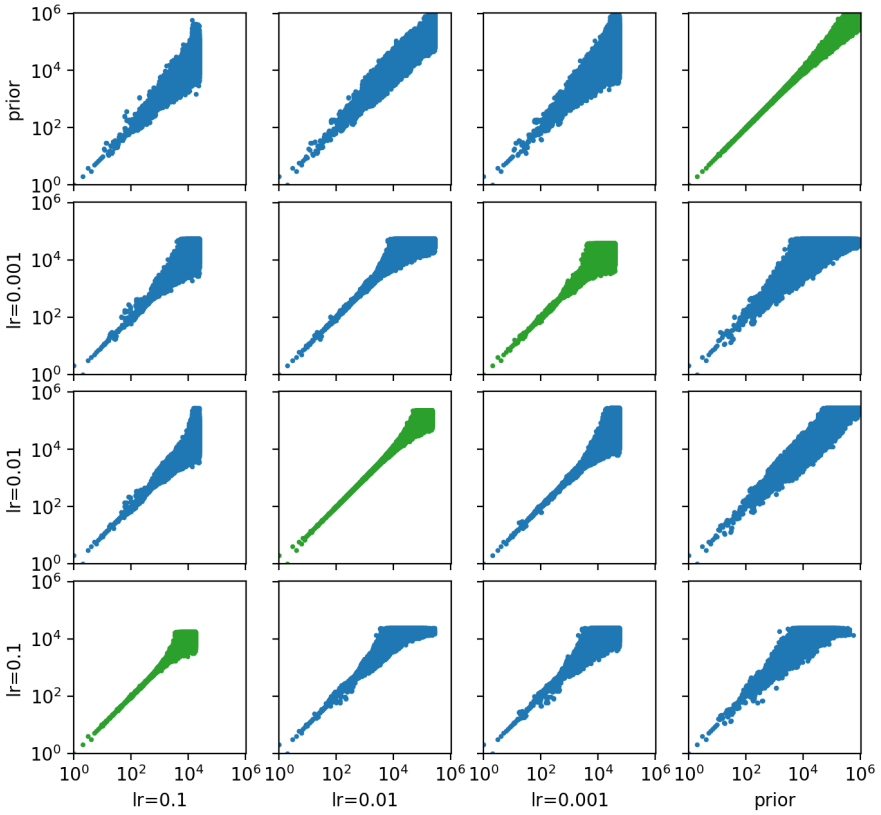
We now extend our analysis from the Bayesian linear model to a single-layer fully connected network (FCN) trained on MNIST using Stochastic Gradient Descent (SGD). Although the probabilistic model is no longer explicit, we can interpret the final function found by the optimiser as a sample from an implicit (non-Bayesian) posterior over functions (see [5]). It is well established that optimiser dynamics can implicitly control regularisation; for example, SGD with a larger learning rate tends to converge to larger margin solutions, which is analogous to reducing λ for fixed σ/ζ [115].

Figure F.3(a–d) shows the effective priors inferred for models trained with both Adam and SGD across a range of learning rates. Consistent with our earlier analogy, increasing the learning rate causes the effective prior to become progressively super-Zipfian, mirroring the effect of decreasing σ^2 in Fig. F.2. However, some of this apparent "super-Zipfian" behaviour may arise from the discrete nature of optimisation with finite step size: very large learning rates can push the optimiser far across the loss landscape, effectively sampling from trivial regions, as commonly observed when networks diverge under extreme step sizes.

Figure F.3(e) further shows that the induced function rankings under these effective priors remain highly—but not perfectly—correlated across optimisers and learning rates. This suggests that different optimisation dynamics imprint distinct



(a) (b) (c) (d)



(e)

Figure F.3: SGD Prior $\tilde{P}(f)$ for different learning rates for a 1-layer FCN on MNIST. (a) shows Adam, $m = 20$, (b) shows SGD, $m = 20$. As the learning rate increases, the exponent of the power law, α, increases. We observe the same pattern in (c) Adam and (d) SGD, where $m = 100$. (e) shows the correlation between the different $\tilde{P}(f)$. The correlation between the methods reduces as the learning rate varies.

inductive biases on the resulting function distributions. In particular, while the general Zipfian structure of the prior persists, each optimiser defines a slightly different effective sampling process over functions, reflecting its own implicit prior.

F.2.4 Concluding remarks

This section reconciles our error-based theoretical framework with the practical reality of training via continuous losses. Using linear models and NNGPs, we demonstrated that loss-based (Gaussian) likelihoods exhibit regimes in which the posterior can collapse onto a single solution by reducing the effective noise scale to zero. We then introduced the *effective prior* as a method for directly comparing the loss-based and error-based likelihoods in our framework. Empirically, when the likelihood is weak (high training error), the effective prior remains Zipfian and matches the initialization prior. As the likelihood becomes strong and training error approaches zero, the effective prior must become super-Zipfian to mimic the excessive concentration of the loss-based posterior—an artifact of forcing an error-based approximation to express certainty it cannot achieve through likelihood alone.

Crucially, this collapse does *not* translate into better generalisation. Consistent with the previous subsection, once zero training error is achieved (already possible under the 0–1 likelihood), further reducing variance by sharpening the loss yields no meaningful test improvement. Moreover, the vanishing-noise regime effectively assumes noise-free data, which is rarely justified in practice. Our analysis in the main text relies on $m_{\text{train}} \to \infty$. For fixed $\lambda > 0$, we showed in this section that the predictive covariance retains an irreducible noise floor, suggesting that 0–1 likelihood may be the strongest likelihood available in the large m_{train} limit.

Finally, extending the analysis to SGD-trained networks, we observed the same qualitative pattern: increasing the learning rate (a stronger effective likelihood) pushes the *effective* prior toward super-Zipfian behaviour, while function rankings remain highly—but not perfectly—correlated across optimisers. This indicates optimiser-specific inductive biases without evidence that the extreme, collapse-inducing regime improves generalisation. In sum, while continuous losses can

suppress posterior variance far beyond what error-based models allow, this suppression reflects overconfidence rather than better out-of-sample performance, and it relies on an unrealistic no-noise assumption.

F.3 Effective Prior Theory

In Appendix F.2 we defined the effective prior informally. Here, we define the effective prior more rigorously, and explain the assumptions we make to compute it.

F.3.1 Nonexponential likelihoods

In this section, we will define two loss-based likelihoods and explain how they can be related to the discrete classifications f. As in the main text, we use the following notation:

- A dataset $D = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, with a training set $S = \{(x_1, y_1), \ldots, (x_{m_{\text{train}}}, y_{m_{\text{train}}})\}$;
- D_X and D_Y denote the set of inputs and targets, respectively. S_X and S_Y denote the training set inputs and targets;
- $f = \{f(x_1), \ldots, f(x_m)\}$ denotes the classification on D, to distinguish it from $f(S_X) = \{f(x_1), \ldots, f(x_{m_{\text{train}}})\}$.

We introduce the following notation for continuous functions:

- $y = \{y_1, \ldots, y_{m_{\text{train}}}\}$ is the value of the targets for the train data;
- $g = \{g(x_1), \ldots, g(x_m)\}$ and $g(S_X) = \{g(x_1), \ldots, g(x_{m_{\text{train}}})\}$ denote the *continuous* output of the neural network on the dataset. Note that $g(x)$ is parameterised by θ (i.e., $g(x; \theta)$) but we often leave the explicit dependence on θ out for simplicity;
- The classification of a network will be done based on the sign of the output. We thus say $f = g$ if $f = \{g(x_1) > 0, \ldots, g(x_m) > 0\}$, and take $P(f|g) = 1$ if $f = g$, else 0.

When it is necessary to specify the f or g produced by some neural-network parameters θ, we write f_θ or g_θ. Consider a neural network with parameters θ and initialisation distribution $P_{\text{par}}(\theta)$. Using our initialisation prior over parameters $P_{\text{par}}(\theta)$, we induce a prior over continuous functions

$$P(g)\,\mathrm{d}g = \delta(g_\theta - g)P_{\text{par}}(\theta)\mathrm{d}\theta.$$

This is related to the initialisation prior via $P(f) = \int \mathrm{d}g\ P(g)P(f|g)$. Abstracting the parameters out in this way is designed to simplify notation in the following sections.

Examples of loss-based likelihoods. In Eq. (F.20), g is assumed to be the output of a softmax activation function (the other two cases use a linear activation). The following three distributions

$$P_{\text{mse}}(f|S) = \frac{\int \mathrm{d}g\ \exp\left(-\frac{\sum_{i=1}^{i=m_{\text{train}}}(g(x_i)-y_i)^2}{2\lambda^2}\right) P(f|g)P(g)}{\sum_{f'}\int \mathrm{d}g'\ \exp\left(-\frac{\sum_{i=1}^{i=m_{\text{train}}}(g'(x_i)-y_i)^2}{2\lambda^2}\right) P(f'|g')P(g')}, \tag{F.19}$$

$$P_{\text{ce}}(f|S) = \frac{\int \mathrm{d}g\ \exp\left(\sum_{i=1}^{i=m_{\text{train}}}\left[y_i \log g(x_i) + (1-y_i)\log(1-g(x_i))\right]\right) P(f|g)P(g)}{\sum_{f'}\int \mathrm{d}g'\ \exp\left(\sum_{i=1}^{i=m_{\text{train}}}\left[y_i \log g'(x_i) + (1-y_i)\log(1-g'(x_i))\right]\right) P(f'|g')P(g')}, \tag{F.20}$$

$$P_{01}(f|S) = \frac{\int \mathrm{d}g\ \mathbb{1}[f(S_X)=y]P(f|g)P(g)}{\sum_{f'}\int \mathrm{d}g\ \mathbb{1}[f'(S_X)=y]P(f'|g)P(g)} = \frac{\mathbb{1}[f(S_X)=y]}{\sum_{f'}\mathbb{1}[f'(S_X)=y]P(f')}P(f) \tag{F.21}$$

are all Bayesian posteriors, and the indices mse, ce, and 01 denote the associated likelihoods, with the mean square error loss, cross-entropy, and a 0–1 error loss in Eqs. (F.19) and (F.20), respectively. For $P_{\text{mse}}(f|S)$, the targets $y_i \in \{-S_T, +S_T\}$ (where the target scale $S_T = 1$ in our experiments), and for $P_{\text{ce}}(f|S)$, $y_i \in \{0, 1\}$.

Because these posteriors have different likelihoods, they may have very different properties. For example, $P_{\text{mse}}(f|S)$ can be tuned with the parameter λ. In the limit of very large λ, $P_{\text{mse}}(f|S) \approx P(f)$ (i.e., we do not learn from the data, but instead randomly sample from the initialisation distribution), and, for small λ, the posterior is very tightly concentrated on a very small part of function-space and more similar to $P_{01}(f|S)$ than $P(f)$ (see [2]). Upon making the target scale S_T large and keeping λ small, this posterior can collapse entirely to a single function f for any given S (see [77] and Appendix F.2 for further discussion).

F.3.2 The effective prior

A naïve approximation of the posterior for some loss-based likelihood $P_l(f|S)$ is to use the initialisation prior $P(f)$ and an exponential likelihood $\exp\{(-\kappa(l)m_{\text{train}}\epsilon\})$, for some suitable choice of $\kappa(l)$, and hope that

$$P_l(f|S) \approx \frac{e^{-\kappa(l)m_{\text{train}}\epsilon}P(f)}{P(S)}. \tag{F.22}$$

If this were a good approximation, we could use our observations that the initialization prior is Zipfian to argue that neural networks are set up optimally for learning. In some trivial cases like a Bayesian neural network using the mse likelihood and very large λ, Eq. (F.22) works well with $\kappa = 0$. (Appendix F.2). However, coarse-graining the real prior $P(g)$ into $P(f)$ before applying a likelihood is a crude approximation, and in other cases, there may be no likelihood that produces posteriors similar to $P_l(f|S)$ (for example, we will see that this is true for mse likelihood with small values of λ). A less naïve approach to approximating $P_l(f|S)$ would be to construct an *effective prior* designed as follows

An effective prior $\tilde{P}(f)$ and associated error-based effective likelihood $\tilde{P}(S|f)$ should accurately approximate the posterior of some underlying learning agent A.

Defining the effective prior

The phrase "accurately approximate" in the previous paragraph hides a lot. The accuracy of the approximation would depend largely on what measure we place on training sets S. Given the real posterior $P_A(f|S)$ that the effective posterior

$$\tilde{P}_A(f|S) = \frac{\tilde{P}(S|f)\tilde{P}(f)}{\tilde{P}(S)}$$

attempts to approximate, a simple quantification of the approximation error would be the KL divergence between the two, averaged over different possible choices of the training set. To calculate the divergence, we subsample S_X of length $m_{\text{train}} = m/2$ from D_X, and then sample S_Y from the initialization prior of the agent A.

Averaging over this distribution of choices of training set S then gives

$$\langle D_{\mathrm{KL}}(\tilde{P}_A(f|S) \parallel P_A(f|S))\rangle_S = \sum_{S_Y} \sum_{\substack{S_X \subset D_X, \\ |S_X|=m/2}} \binom{m}{m/2}^{-1} P_A(S_Y|S_X) \sum_f P_A(f|S) \log\left(\frac{P_A(f|S)}{\tilde{P}_A(f|S)}\right), \tag{F.23}$$

where $P_A(D_Y|D_X)$ is the initialisation prior. $\binom{m}{m/2}^{-1}$ is the probability of sampling any input set S_X of size $m/2$, and $P_A(S_Y|S_X)$ is the initialisation prior probability of assigning labels S_Y to S_X. The effective prior–effective likelihood combination would minimise $\langle D_{\mathrm{KL}}(\tilde{P}_A(f|S) \parallel P_A(f|S))\rangle_S$.

Can we compute $\tilde{P}$?

Once we have the *effective prior* $\tilde{P}(f)$, we can reason about the behaviour of the underlying learning agent using the arguments in the rest of this paper. However, computing $\tilde{P}(f)$ by minimizing Eq. (F.23) would be very computationally expensive, and we do not attempt it. Instead, in the rest of this Appendix, we will introduce a method based on Bayesian sequential updating for $\tilde{P}$, which we hope approximately minimizes Eq. (F.23).

In Appendix F.3.2, we introduce this method and show that the effective prior is the prior for error-based likelihoods. In Appendix F.3.2 we discuss how this method deviates from the ideal approximation for loss-based likelihoods. Finally, in Appendix F.3.2, we discuss the errors of this approximation for non-Bayesian learning agents.

Building up priors for error-based likelihoods

In this section, we study error-based likelihoods exclusively—meaning the effective prior *is* the prior. We do this to introduce a method of sequentially adding data, which will be used approximately in the next section to construct the effective priors for loss-based likelihoods. We will consider the binary classification case for simplicity, but note that this method can be extended to the multiclass case straightforwardly. Given some data S_X, we will use the following notation

- $f^k = \{f(x_1), \ldots, f(x_k)\}$ is the restriction of f to the first k datapoints in D.

We begin by assuming that we have a Bayesian prior $P(f)$ and error-based likelihood $P(S|f)$. Bayesian sequential updating is the process of refining a posterior by sequentially adding data. We use a similar procedure to compute the prior $P(f^{k+1})$ given $P(f^k)$ given in Algorithm 10.

Algorithm 10 Exact Priors from Sequential Sampling

Require: Learning agent A, data points $D_X = \{x_1, \ldots, x_m\}$.

1: Initialize $\hat{P}(f^1)$ by sampling $f(x_1)$ from agent A.
2: **for** $k = 2, \ldots, m$ **do**
3: **for** each function f'^{k-1} with $\hat{P}(f'^{k-1}) > 0$ **do**
4: Estimate the conditional distribution $\hat{P}(\cdot|f'^{k-1})$ by "training" A and sampling.
5: **end for**
6: Compute the marginal likelihood

$$\hat{\mu}(f^k) = \sum_{f'^k} P(f^k|f'^k)\hat{P}(f'^k)$$

 where for exponential likelihoods, $P(f^k|f'^k) = e^{-\kappa\epsilon(f,f')}$.
7: Compute the marginal distribution $\hat{P}(f^k)$ for each possible function f^k using the law of total probability:

$$\hat{P}(f^k) \leftarrow \sum_{f'^{k-1}} \hat{P}(f^k|f'^{k-1})\hat{\mu}(f'^{k-1})$$

8: **end for**
9: **Output:** The final distribution $\hat{P}(f^m)$.

Algorithm 10 exactly computes $P(f)$, but is much slower and more inefficient than just sampling over the entire dataset. We will show how it can be simplified and approximated for the 01 likelihood and exponential likelihood in the remainder of this section.

01 likelihood For the 01 likelihood ($\kappa \to \infty$), where $P(f^{k+1}|f'^k) = \mathbb{1}[f^k = f'^k]$, we can simplify the above. In this case, $\mu(f^k) = P(f^k)$, and the following holds

$$P(f^k) = P(f^k|f^{k-1})P(f^{k-1}), \tag{F.24}$$

$$P(f^k) = P(f^k|f^{k-1}) \ldots P(f^2|f^1)P(f^1). \tag{F.25}$$

Therefore, if we want to build up $P(f^k)$, we can continue to use Algorithm 10, but it is easier to use Algorithm 11.

If $P(f'^{k+1}|f)$ is the probability of randomly initializing to f'^{k+1} given that f'^{k+1} is consistent with f^k, Algorithm 11 would generate the initialization prior.

Algorithm 11 Approximate prior from sampling

Require: Learning agent A, Samples $\mathtt{S} \leftarrow \{\}$
1: **for** n_s **do**
2: $f^1 \sim P(x_1)$
3: **for** $k = 1$ to $k = m - 1$ **do**
4: $f' \sim P(f'^{k-1}|f^k)$
5: $f \leftarrow f[:k] + f'[k+1]$ {Append the new bit to f}
6: **end for**
7: Append f to $\mathtt{S}$
8: **end for**

Exponential likelihood For $P(S|f) = e^{-\kappa m_{\text{train}}\epsilon}$, we use Algorithm 10, as the posterior is easy to compute and sample from when using GPs.

Building up priors for loss-based likelihoods

As discussed in Appendix F.2, once we use loss-based likelihoods, the effective prior would no longer necessarily equal to the prior. This means posteriors created from Algorithm 10 would no longer be exactly the same as the true posterior $P_l(f|S)$. What about the effective likelihood? A complete analysis would fit some error-based likelihood to $\tilde{P}_l(S|f)$. We assume the following.

1. When the number of errors made on the training set is close to zero (we chose $< 2\%$), we use 01 likelihood, as no "stronger" error-based likelihood exists. Once we make this assumption, we use Algorithm 11.

2. When the number of errors made on the training set is nonzero (in the 10^6 or more samples taken) we assume that the continuous likelihood, found implicitly in $P(f^k|f'^{k-1})$ can be well approximated by some function of error only.

Future work would investigate both of these assumptions further.

Building up priors for SGD

For non-Bayesian SGD posteriors, we can use the same approximations. We can use Algorithm 11 if we train to 100% training accuracy at each step. All the errors in the approximations remain from Appendix F.3.2, with the addition that SGD is non-Bayesian. A key point to note is that details of the stopping condition and optimizer will affect the form of $\tilde{P}_{SGD}(f)$. For example, larger learning rates will penetrate further across the decision boundary, leaving the stopping condition less precise. Furthermore, the batch size is not constant until $k =$ `batch_size`. In our experiments, we will sometimes use a batch size greater than m (which would effectively be a GD prior), and sometimes use batch size 32, with $m = 100$ (which would be more like an SGD prior). However, we find very little difference between the SGD and GD priors at this scale.

Summary of results

Our numerical results are summarized in Table F.3.

F.3.3 Pseudocode for computing the effective SGD prior

In the case of optimizer-trained neural networks, the posterior $P_{SGD}(f|S)$ is not Bayesian. We train to 0 training error, meaning we can use Algorithm 11 (see Appendix F.3.2 for a discussion). See the pseudocode below.

Likelihood	Condition	Eff. prior approximation	Eff. prior behaviour
Error-based likelihood	Exponential	Algorithm 1, exact	Zipf
	Zero-one	Algorithm 2, exact	Zipf
Loss-based likelihood	$\epsilon_{\text{train}} > 0$	Algorithm 1, approx	Zipf
	$\epsilon_{\text{train}} = 0$	Algorithm 2, approx	Zipf/Super-Zipf
SGD	$\epsilon_{\text{train}} = 0$	Algorithm 2, approx	Zipf/Super-Zipf

Table F.3: Comparison of different algorithms and their behavior

```
# Assume binary classification task
from collections import Counter
import Data # input data
import DNN # deep neural network, bias terms disabled
import Optim # optimiser
import train # trains network to fit target, stops at 0 train error
import shuffle # shuffles data

def sample_function(network, S, optim, batch_size):
    shuffled_S = shuffle(S)
    y = [random.randint(0, 1)]
    for idx in range(1, m):
        network.initialise_parameters()
        # init params from some initialisation scheme
        train(
            network, shuffled_S[:idx], y[:idx],
            optim, min(batch_size,idx)
        )
        y.append(network.evaluate(shuffled_S[idx]))
    f = network.evaluate(S)
    return f

S = Data()
network = DNN()
optim = Optim()
batch_size = 100

f_frequency = []
for count in range(total_samples):
    f_frequency.append(
        sample_function(network, S, optim, batch_size)
    )
# dictionary of frequency of functions
Prior_f = dict(Counter(f_frequency))
# dictionary of \p{f}
Prior_f = {i:j/total_samples for i, j in Prior_f.items()}
```

F.4 Supplementary experiments

This appendix shows supplementary experiments which (1) show the robustness of our results, and (2) answer some secondary questions they may raise.

F.4.1 Robustness of Zipf's law to architecture and dataset choices

Firstly, we show supplementary experiments to Fig. 5.1 in the main text, demonstrating Zipf's law across a wider range of architectures and datasets. The panels in Fig. F.4 show the following:

- **Invariance to dataset:** Panels (a) and (b) confirm that a 3-layer Fully Connected Network (FCN) produces a Zipfian prior for both the MNIST and CIFAR-10 datasets. Furthermore, panel (c) shows the law holds for a 5-class CIFAR-10 task, indicating that the phenomenon is not exclusive to binary classification problems.

- **Invariance to network width:** Panel (d) demonstrates that the prior distribution remains Zipfian across networks of varying widths.

- **Robustness to initialisation scale:** Panel (e) investigates the role of the bias initialisation scale, σ_b. While this hyperparameter strongly influences the probability of trivial (zero-information) and other low-entropy functions, the overall Zipfian trend for the remaining functions holds.

- **Zipf's law when $m > d$:** Finally, panel (f) addresses the relationship between the number of data points (m) and the input dimension (d). It shows that Zipf's law can still emerge when the number of data points exceeds the input dimension (here, $m = 100$ and $d = 8$), unlike most cases in the main text where $d > m$.

F.4.2 Robustness of power-law fits to lower cutoff rank

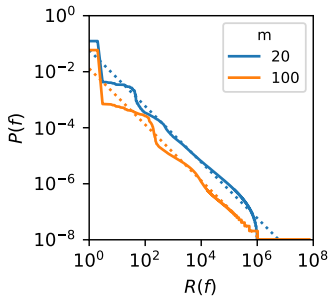
Recall that the first few functions generally have probabilities which deviate from Zipf's law, and thus we fit distributions to a normalized power-law distribution starting from some initial cutoff rank s (Appendix F.1.3) Fig. F.5 shows that the Zipf's law fit (using Eq. (F.4)) is insensitive to its starting point, s (cf. Appendix F.1.3).

F.4.3 Architecture dependence of function ordering

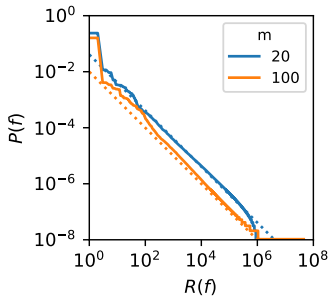
Fig. 5.1(b) and (f) demonstrate that different architectures on the same dataset exhibit Zipf's law. However, these architectures are known to have very different inductive biases (as trained ResNets outperform FCNs on CIFAR-10 by tens of percentage points). This would suggest that the ordering of functions in the prior should be significantly different. Fig. F.6 clearly demonstrates this. It compares the function priors from an FCN, a CNN, and a ResNet on CIFAR-10. Each off-diagonal plot compares the rank of functions between two different architectures' priors, while the diagonal plots show the noise scale by comparing two independent samples from the same prior. As expected, we see significant differences in the ranks for different architectures.

F.4.4 Subzipfian priors on boolean data

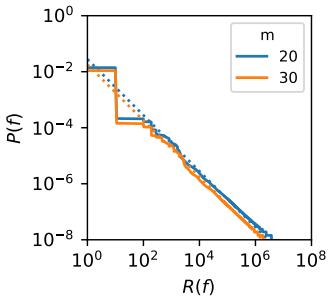
In the main text, Fig. 5.2, we showed that FCNs can produce sub-Zipfian priors in the chaotic regime. In Fig. F.7 we show how a perceptron on the Boolean dataset can also generate a sub-Zipfian prior. However, this is not a property of the Boolean data (as (b) shows Zipf's law for an FCN on Boolean data), nor the perceptron (as (e) shows the perceptron on Fashion MNIST satisfies Zipf's law), nor is it due to the perceptron having fewer parameters than datapoints (see a counterexample in (f)). Instead, arguments in Appendix F.8.3 imply that it is a property of the kernel (c)—namely, that kernels with a lot of small off-diagonal elements should be sub-Zipfian. In support of this interpretation, we show a GP with the perceptron kernel ($K(x, x') = \sigma_w^2 x \cdot x' + \sigma_b^2$; we use $\sigma_w = 1$ and $\sigma_b = 0.01$) in (d)—with very similar results to (a).



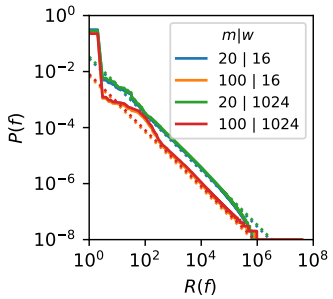
(a) FCN on MNIST



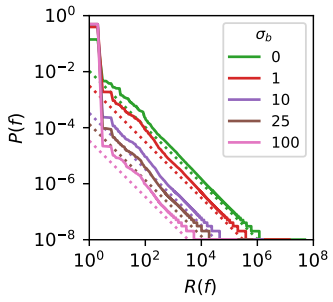
(b) FCN on CIFAR-10



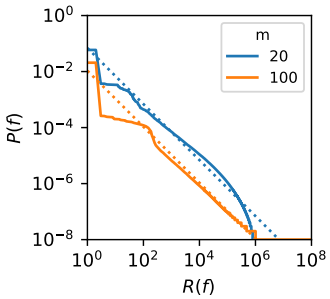
(c) FCN on 5 class CIFAR-10



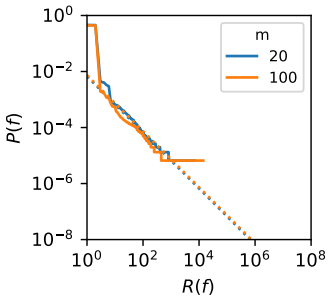
(d) FCN on Fashion MNIST



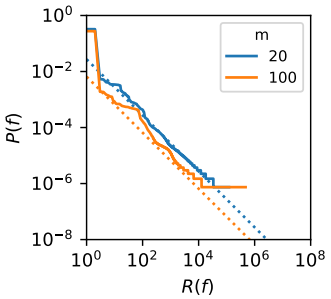
(e) FCN on MNIST, effect of σ_b



(f) input dimension $d = 8$



(g) Transfomer (fully random)



(h) LSTM

Figure F.4: Supplementary architectures and datasets. All architectures are 3-layer FCNs of width 128 unless otherwise specified. (a) shows the FCN on MNIST, complementing the datasets shown in Fig. 5.1(a),(e). (b) shows the FCN on CIFAR-10, complementing the architectures shown in Fig. 5.1(b),(f). (c) shows the FCN on CIFAR-10 but with 5 classes, demonstrating that Zipf's law is not unique to binary classification. (d) shows Zipf's law for the same 3-layer FCN on Fashion MNIST but for different widths, and (e) varies the bias term initialization (all experiments use $m = 100$), showing that it strongly affects the probability of the low-rank functions, but does not disrupt Zipf's law. (f) shows that Zipf's law emerges even with more datapoints $m = 100$ than input dimension $d = 8$ (datapoints sampled from an i.i.d. Gaussian). (g) shows the same transformer as in Fig. 5.1(c) on the IMDb dataset, but the entire network is randomly initialized (as opposed to just the last layer). (h) shows an LSTM on the IMDb dataset. Due to the substantial computational cost associated with their large number of parameters, both models were sampled fewer than 10^8 times.

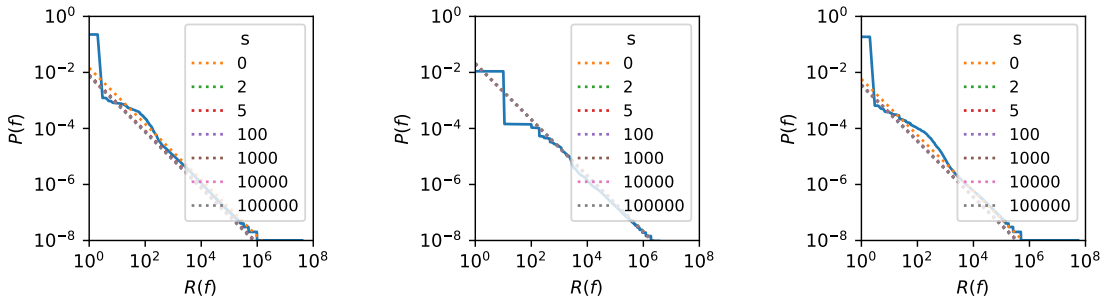


Figure F.5: Robustness of Zipfian fit to choice of low-rank cutoff. We test Eq. (F.4) with different values of s (the starting rank used for the Zipf's law fit) for a 3-layer FCN (a) on Fashion MNIST with $m = 100$, (b) on 5-class CIFAR-10 with $m = 30$ and (c) Fashion MNIST with $m = 500$. The fit (shown by the dotted lines) is not strongly dependent on s.

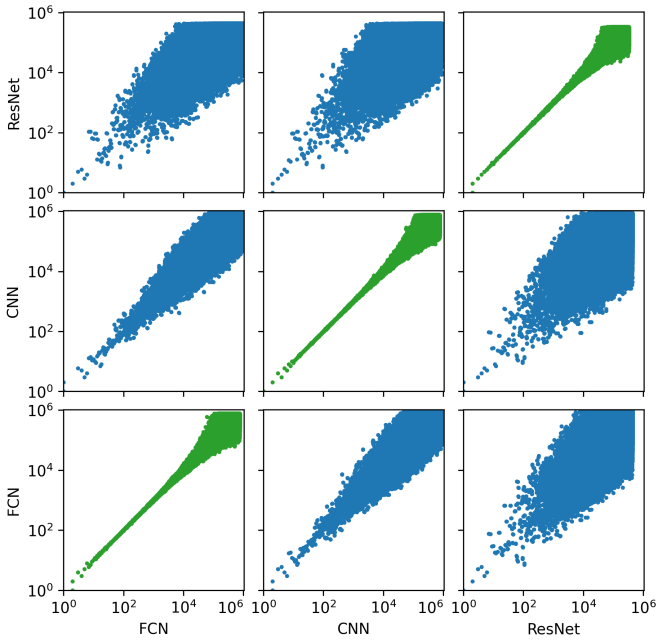
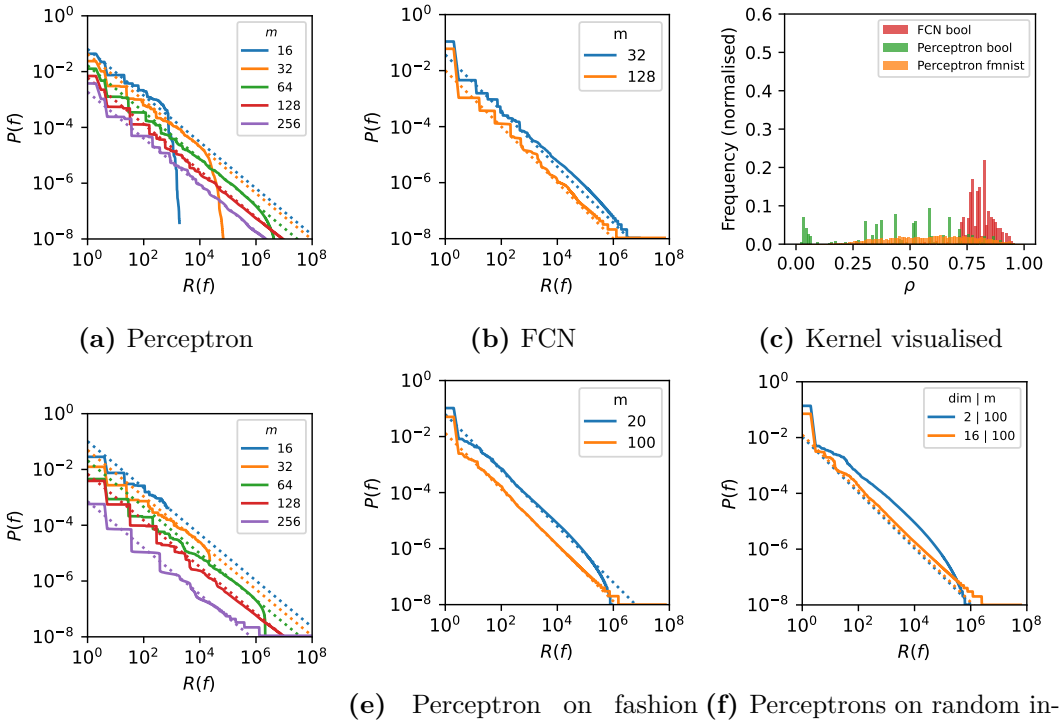


Figure F.6: Effect of architecture on the ordering of functions in the prior. We use CIFAR-10, $m = 20$, with data from Fig. 5.1(b),(f) and Fig. F.4b. The plots on the diagonal show two different samples of size 5×10^7 taken from the same prior, to show the noise scale. The off-diagonal plots compare the priors of two different networks (with 10^8 samples). Each datapoint is a function. The x and y coordinates show the Rank of f for the two samples—for example, in the top right plot, the x-coordinate is Rank(f) for the ResNet, and the y-coordinate is Rank(f) for the FCN. Differences in the architectures' inductive biases are revealed by the differences in function ranking Rank(f). Clearly, there are significant differences between architectures, as in all cases the deviation from $y = x$ is far greater than can be explained by the noise scale.



(a) Perceptron **(b)** FCN **(c)** Kernel visualised

(d) GP (perceptron kernel) **(e)** Perceptron on fashion MNIST **(f)** Perceptrons on random input data

Figure F.7: Perceptrons on Boolean & Other Data. Here, we show another learning agent–dataset pair that is sub-Zipfian. The weights and biases were sampled from i.i.d. Gaussians with the variance of the bias term $\sigma_b^2 = \sigma_w^2/d$, where d is the perceptron dimension and $\sigma_w = 1$. (a) a perceptron on the Boolean dataset with inputs $x \in \{0,1\}^n$ for $n = 4, 5, 6, 7, 8$. $n = 7$ has 10^9 samples, all others have 10^8. Power law fits show the $n = 7$ and $n = 8$ ($m = 128$ and $m = 256$ as $m = 2^n$) plots to have best estimates of $\alpha = 0.83$ and $\alpha = 0.84$ respectively. (b) Shows an FCN for $n = 5, 7$, which is Zipfian. (c) shows the histograms of their kernels, suggesting that the perceptron is sub-zipfian due to a large fraction of $\rho_{ij} < 0.5$, unlike the 3-layer FCN (see Eq. (F.142) for an explanation). (d) shows a GP with a perceptron kernel, which has almost identical performance to (a). (e) shows the perceptron on Fashion MNIST (which is Zipfian), see (c) for its kernel. (f) shows two perceptrons, one with input dimension 2 and one with dimension 16 acting on 100 datapoints (randomly sampled from i.i.d. Gaussians). Note that these perceptrons are not fully expressive, unlike most architectures shown in the main text, but still show Zipf's law.

F.5 Derivation of lower bounds on the generalisation error

This appendix presents the derivation of lower bounds on the generalisation error for priors with different power-law scalings. We use tools from statistical physics, particularly large deviation theory, to make our arguments. The results derived here provide the theoretical foundation for the claims made in Section 5.3 of the main text.

Suppose the input space consists of m possible inputs, of which we train on $m_{\text{train}} = qm$. We compute the test error on the $(1-q)\,m$ unobserved inputs, i.e., the "off training set" error [83]. We define each function $f^{(\nu)}$ by its classification of the full space of m inputs, and then denote its prior probability by $P_\nu \equiv P(f^{(\nu)})$. Given this prior P_ν, we perform Bayesian inference. Defining the per sample training set error $\epsilon_S[\nu]$ and test set error $\epsilon_G[\nu]$, we consider the likelihood

$$L_\nu = e^{-\kappa qm\epsilon_S[\nu]}. \tag{F.26}$$

As discussed in Section 5.3, this is the likelihood of the labels, given $f^{(\nu)}$, if examples are mislabelled independently with probability $1/(1+e^\kappa)$. We compute the posterior $Q_\nu \equiv P(f^{(\nu)}|S)$ through an application of Bayes' rule

$$Q_\nu = \frac{P_\nu L_\nu}{\sum_\nu P_\nu L_\nu} \equiv \frac{1}{Z}\tilde{Q}_\nu, \tag{F.27}$$

where we defined the unnormalized "posterior weight" $\tilde{Q}_\nu = P_\nu L_\nu$. The model samples from the posterior to make predictions on the $(1-q)\,m$ unseen inputs.

We assume a power-law prior of the form $P_\nu \sim 1/r_\nu^\alpha$. This still leaves the *ordering* of functions in the prior unspecified. In order to prove *lower bounds* on the test error, we will choose the ordering to provide the strongest possible inductive bias for learning functions with low error relative to the target function f^*. Any lower bounds on the test error in this setting must also hold with less favourable priors.

This prior has the function f^* as the most likely function in the prior, i.e. rank $f^* = 1$. Every other function $f^{(\nu)}$ differs from f^* on $\epsilon_\nu m$ data points, where $\epsilon_\nu = q\epsilon_S[\nu] + (1-q)\,\epsilon_G[\nu]$. The prior is optimal when rank r_ν is a monotonically increasing function of ϵ_ν.

Our results will make use of $\tilde{Q}(\epsilon)$, the total posterior weight of all functions with error rate ϵ, given by

$$\tilde{Q}(\epsilon) \equiv \sum_{\{\nu:\epsilon_\nu=\epsilon\}} \tilde{Q}_\nu = \sum_{\{\nu:\epsilon_\nu=\epsilon\}} P_\nu L_\nu, \tag{F.28}$$

To compute this, we first have to compute the value of P_ν for all functions $f^{(\nu)}$ with $\epsilon_\nu = \epsilon$. Crucially, all of these functions have essentially the same P_ν in the large-deviation sense. Since the number of functions with exactly k errors is $\binom{m}{k}$, our optimistic ordering of functions in the prior gives

$$\frac{1}{\sqrt{8m\left(\epsilon_\nu - \frac{1}{m}\right)\left(1 + \frac{1}{m} - \epsilon_\nu\right)}} 2^{mH_2\left(\epsilon_\nu - \frac{1}{m}\right)} \leq \sum_{k=0}^{m\epsilon_\nu - 1} \binom{m}{k} \leq r_\nu \leq \sum_{k=0}^{m\epsilon} \binom{m}{k} \leq 2^{mH_2(\epsilon_\nu)}, \tag{F.29}$$

where $H_2(\cdot)$ is binary entropy such that $H_2(x) = -x\log_2 x - (1-x)\log_2(1-x)$. This means that

$$\frac{1}{\sqrt{8m\left(\epsilon - \frac{1}{m}\right)\left(1 + \frac{1}{m} - \epsilon\right)}} 2^{m\alpha H_2\left(\epsilon - \frac{1}{m}\right)} \sum_{\{\nu:\epsilon_\nu=\epsilon\}} L_\nu \geq \tilde{Q}(\epsilon) \geq 2^{m\alpha H_2(\epsilon)} \sum_{\{\nu:\epsilon_\nu=\epsilon\}} L_\nu. \tag{F.30}$$

For large m, the upper and lower bounds on $\frac{1}{m}\log r$ converge, giving the large-deviation form [260]

$$\tilde{Q}(\epsilon) \asymp 2^{-m\alpha H_2(\epsilon)} \sum_{\{\nu:\epsilon_\nu=\epsilon\}} L_\nu. \tag{F.31}$$

With the preliminaries now dispensed with, we can begin presenting our results. We will first consider the case where $\kappa \to \infty$ (the hard likelihood) as the proof is more straightforward, and this result is more relevant to overparameterised neural networks that achieve zero training error.

Theorem 7. (Hard likelihood) *Bayesian learning algorithms with a power-law prior* $P_\nu \propto 1/\operatorname{rank} f^{(\nu)^\alpha}$ *and hard likelihood function* $L_\nu = \begin{cases} 1 & \text{if } \epsilon_S[\nu] = 0 \\ 0 & \text{otherwise} \end{cases}$ *will have test error* $\epsilon_G[\nu] > 0$ *as the amount of data* $m \to \infty$ *for* $\alpha < 1$.

Proof. When the likelihood is always 0 or 1, the sum of the likelihoods simply counts the number of functions with error ϵ that agree with the training data. A

function with error rate ϵ agrees with the training data if all ϵm errors are located in the test set (of size $(1-q)m$). The sum of the likelihoods is then simply

$$\sum_{\{\nu:\epsilon_\nu=\epsilon\}} L_\nu = \binom{(1-q)m}{\epsilon m} \asymp 2^{m(1-q)H_2\left(\frac{\epsilon}{1-q}\right)}. \tag{F.32}$$

The posterior weight of these functions will therefore be

$$\tilde{Q}(\epsilon) = 2^{-m\alpha H_2(\epsilon)} \sum_{\{\nu:\epsilon_\nu=\epsilon\}} L_\nu \asymp 2^{-mI_{\text{hard}}(\epsilon|q,\alpha)}, \tag{F.33}$$

where

$$I_{\text{hard}}(\epsilon|q,\alpha) = \alpha H_2(\epsilon) - (1-q)\, H_2\left(\frac{\epsilon}{1-q}\right). \tag{F.34}$$

By direct calculation of the second derivative, we find that this is a convex function of ϵ for $\alpha, q < 1$ and $\epsilon \in [0, 1-q]$ (which must be true as all errors occur on the test set). Thus, as $m \to \infty$, samples from the posterior will have $\epsilon = \epsilon^*$ almost surely, where ϵ^* is the minimizer of $I_{\text{hard}}(\epsilon|q,\alpha)$. We minimise by differentiating with respect to ϵ, and then substitute $\epsilon_G = \epsilon/(1-q)$ (as the training error is zero), giving

$$\frac{\log\left(\frac{1}{\epsilon_G^*} - 1\right)}{\log\left(\frac{1}{(1-q)\epsilon_G^*} - 1\right)} = \alpha. \tag{F.35}$$

This equation has a nonzero solution for any $\alpha, q < 1$, and so, as $m \to \infty$, $\epsilon_G = \epsilon_G^* > 0$. □

Eq. (F.35) can only be solved explicitly for ϵ_G^* for certain rational values of α, but the function $\epsilon_G^*(\alpha, q)$ can be constructed implicitly by solving for α or q. Curves $\epsilon_G^*(\alpha, q)$ are shown in Fig. F.8, showing that $\epsilon_G^* > 0$ for all $\alpha, q < 1$.

For $\alpha \geq 1$, Eq. (F.35) has no solutions, and the minimum of Eq. (F.34) occurs at the boundary $\epsilon_G = 0$, so we do not achieve a non-trivial lower bound.

In the remainder of this section, we will refer to ϵ_G^* for the hard likelihood as $\epsilon_G^*(\kappa \to \infty)$.

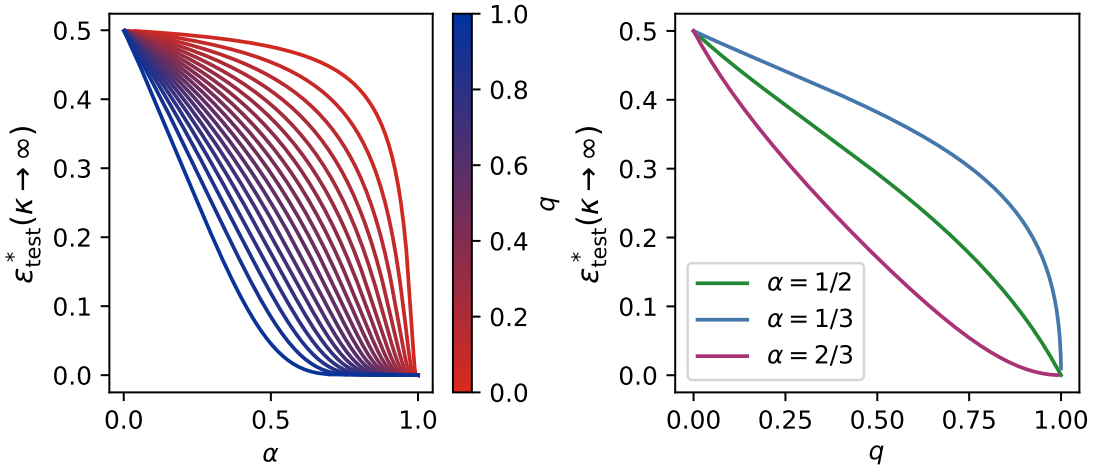


Figure F.8: Solutions ϵ^*_G to Eq. (F.35). (A) $\epsilon^*_G(\alpha)$ for $q = 0.05, 0.1, \ldots, 0.9, 0.95$ (red to blue). At fixed q, all values of $\alpha < 1$ give nonzero ϵ^*_G, and ϵ^*_G increases as α is decreased. (B) $\epsilon^*_G(q)$ for indicated values of α. In all cases, ϵ^*_G ranges from 0 to 1/2 as q varies from 0 to 1.

Theorem 8. (Exponential likelihood) *Bayesian learning algorithms with a power-law prior* $P_\nu \propto 1/\operatorname{rank} f^{(\nu)^\alpha}$ *and exponential likelihood function* $L_\nu = e^{-\kappa q m \epsilon_S[\nu]}$ *will have test error* $\epsilon_G[\nu] > 0$ *as the amount of data* $m \to \infty$ *for* $\alpha < 1$.

Proof. Again we wish to compute the function $\tilde{Q}(\epsilon)$, but it is no longer the case that only functions with $\epsilon_S = 0$ contribute to the posterior. Instead, we can compute the sum of likelihoods of functions with the same ϵ by counting the ways the errors can be split between the training and test set:

$$\sum_{\{\nu:\epsilon_\nu=\epsilon\}} L_\nu = \sum_{qm\epsilon_S=0}^{qm} \binom{qm}{qm\epsilon_S} \binom{(1-q)\,m}{(\epsilon - q\epsilon_S)\,m} e^{-\kappa q m \epsilon_S} \tag{F.36}$$

For large m we can find the large deviation form of the binomial coefficients using Stirling's approximation, and then use the saddle point method [260] to approximate this sum, giving

$$\sum_{\{\nu:\epsilon_\nu=\epsilon\}} L_\nu \asymp \sum_{qm\epsilon_S=1}^{qm} 2^{qmH_2(\epsilon_S)+(1-q)mH_2\left(\frac{\epsilon-q\epsilon_S}{1-q}\right)} \cdot e^{-m\kappa q\epsilon_S} \tag{F.37}$$

$$\asymp 2^{-mF\left(\epsilon^*_S|\epsilon,\kappa,q\right)}, \tag{F.38}$$

where ϵ_S^* is the minimizer of

$$F(\epsilon_S|\epsilon,\kappa,q) = -qH_2(\epsilon_S) - (1-q)\,H_2\left(\frac{\epsilon - q\epsilon_S}{1-q}\right) + \kappa q\epsilon_S \log_2 e\,. \tag{F.39}$$

This is a sum of convex functions of ϵ_S and is thus a convex function of ϵ_S, and has a single minimum $\epsilon_S^*(\epsilon)$ as required. Therefore, the posterior weight on functions of error ϵ has the large-deviation form

$$\tilde{Q}(\epsilon) = 2^{-m\alpha H_2(\epsilon)} \sum_{\{\nu:\epsilon_\nu=\epsilon\}} L_\nu \asymp 2^{-m\alpha H_2(\epsilon)} \cdot 2^{-mF\left(\epsilon_S^*|\epsilon,\kappa,q\right)} \equiv 2^{-mI(\epsilon|\kappa,q,\alpha)}, \tag{F.40}$$

where

$$I(\epsilon|\kappa,q,\alpha) = \alpha H_2(\epsilon) - (1-q)\,H_2\left(\frac{\epsilon - q\epsilon_S^*}{1-q}\right) - qH_2(\epsilon_S^*) + \kappa q\epsilon_S^* \log_2 e. \tag{F.41}$$

The rate function $I(\epsilon|\kappa,q,\alpha)$ is a convex function of ϵ, since

$$\frac{\mathrm{d}^2 I}{\mathrm{d}\epsilon^2} = \alpha\frac{\mathrm{d}^2 H_2(\epsilon)}{\mathrm{d}\epsilon^2} + \frac{\partial^2 F}{\partial\epsilon^2} + \frac{\partial\epsilon_S^*}{\partial\epsilon}\left.\frac{\partial^2 F}{\partial\epsilon\partial\epsilon_S}\right|_{\epsilon_S^*} + \frac{\mathrm{d}}{\mathrm{d}\epsilon}\left(\frac{\partial\epsilon_S^*}{\partial\epsilon}\left.\frac{\partial F}{\partial\epsilon_S}\right|_{\epsilon_S^*}\right) \tag{F.42}$$

$$= \frac{\partial^2}{\partial\epsilon^2}\left(\alpha H_2(\epsilon) - (1-q)H_2\left(\frac{\epsilon - q\epsilon_S^*}{1-q}\right)\right) \tag{F.43}$$

$$\geq \frac{\partial^2}{\partial\epsilon^2}\left(\alpha H_2(\epsilon) - (1-q)H_2\left(\frac{\epsilon}{1-q}\right)\right) = \frac{\partial^2 I_{\text{hard}}}{\partial\epsilon^2} > 0, \tag{F.44}$$

for $\alpha < 1$ and other variables restricted to their domains.

Therefore as $m \to \infty$ samples from the posterior will have $\epsilon = \epsilon^*$ almost surely, where ϵ^* minimizes $I(\epsilon|\kappa,q,\alpha)$.

To fully determine the rate function, we have to solve the saddle point equation for ϵ_S^*:

$$\left.\frac{\partial F(\epsilon_S|\epsilon,\kappa,q)}{\partial\epsilon_S}\right|_{\epsilon_S=\epsilon_S^*} = 0 \tag{F.45}$$

$$\log\left(\frac{1}{\epsilon_S^*} - 1\right) - \log\left(\frac{1-q}{1-\epsilon-(1-\epsilon_S^*)\,q} - 1\right) = \kappa \tag{F.46}$$

$$\frac{-e^\kappa - q + e^\kappa q - \epsilon + e^\kappa\epsilon + \sqrt{4(e^\kappa-1)q\epsilon + (e^\kappa + q - e^\kappa q + \epsilon - e^\kappa\epsilon)^2}}{2(-1+e^\kappa)q} = \epsilon_S^*. \tag{F.47}$$

While we can combine Eqs. (F.41) and (F.47) to get an explicit equation for $I(\epsilon|\kappa, q, \alpha)$, finding the saddle point ϵ^* analytically can only be done in certain simplifying limits.

In the limit as $\kappa \to \infty$, $\epsilon_S^* \to 0$. It's easy to see from Eqs. (F.34) and (F.41) that

$$\lim_{\kappa\to\infty} I(\epsilon|\kappa, q, \alpha) = I_{\text{hard}}(\epsilon|q, \alpha), \tag{F.48}$$

and we recover the results of the hard likelihood analysis above, achieving the same $\epsilon_G^*(\kappa \to \infty) > 0$ as before.

For finite κ, the saddle point is hard to find analytically. We will instead establish that the test error is nonzero by bounding ϵ_G^* with respect to $\epsilon_G^*(\kappa \to \infty)$. First, we show that, for a fixed ϵ, q, α, $\mathrm{d}I/\mathrm{d}\epsilon$ is a strictly increasing function of κ, as

$$\frac{\mathrm{d}}{\mathrm{d}\kappa}\frac{\mathrm{d}I}{\mathrm{d}\epsilon} = \frac{\mathrm{d}^2 I}{\mathrm{d}\epsilon\,\mathrm{d}\kappa} = \frac{\mathrm{d}}{\mathrm{d}\epsilon}\left(\frac{\partial F}{\partial \kappa} + \frac{\partial F}{\partial \epsilon_S}\frac{\mathrm{d}\epsilon_S^*}{\mathrm{d}\kappa}\right) \tag{F.49}$$

$$= \underbrace{\frac{\partial^2 F}{\partial\kappa\,\partial\epsilon}}_{=0} + \underbrace{\frac{\partial F}{\partial\kappa\,\partial\epsilon_S}}_{>0}\underbrace{\frac{\mathrm{d}\epsilon_S^*}{\mathrm{d}\epsilon}}_{>0} + \frac{\mathrm{d}\epsilon_S^*}{\mathrm{d}\kappa}\underbrace{\frac{\mathrm{d}}{\mathrm{d}\epsilon}\frac{\partial F}{\partial\epsilon_S}}_{=0} + \underbrace{\frac{\partial F}{\partial\epsilon_S}}_{=0}\frac{\mathrm{d}^2\epsilon_S^*}{\mathrm{d}\kappa\mathrm{d}\epsilon} \tag{F.50}$$

$$> 0. \tag{F.51}$$

We also have that I is a convex function of ϵ, and that $\mathrm{d}I/\mathrm{d}\epsilon < 0$ at $\epsilon = 0$ (from the analysis of $\epsilon_S = 0$ case). The minimum of I is found at ϵ^* where $\mathrm{d}I/\mathrm{d}\epsilon$ goes from negative to positive, and Eq. (F.51) tells us that this occurs sooner as κ increases. Therefore ϵ^* is a strictly decreasing function of κ, and we have that $\epsilon^* \geq \epsilon^*(\kappa \to \infty)$.

Further, as the likelihood prefers functions with lower values of ϵ_S at fixed ϵ, we have that $\epsilon_G^* \geq \epsilon^*$.

This gives us a final bound of $\epsilon_G^* \geq (1-q)\epsilon_G^*(\kappa \to \infty)$ for $\alpha, q < 1$. Therefore, as $m \to \infty$, for any $\alpha < 1$ and any κ, $\epsilon_G = \epsilon_G^* > 0$. □

Numerical solutions of the saddle-point equations at finite κ are shown in Fig. F.9, showing that the lower bound on the test error is indeed larger than that for the hard (0–1) likelihood.

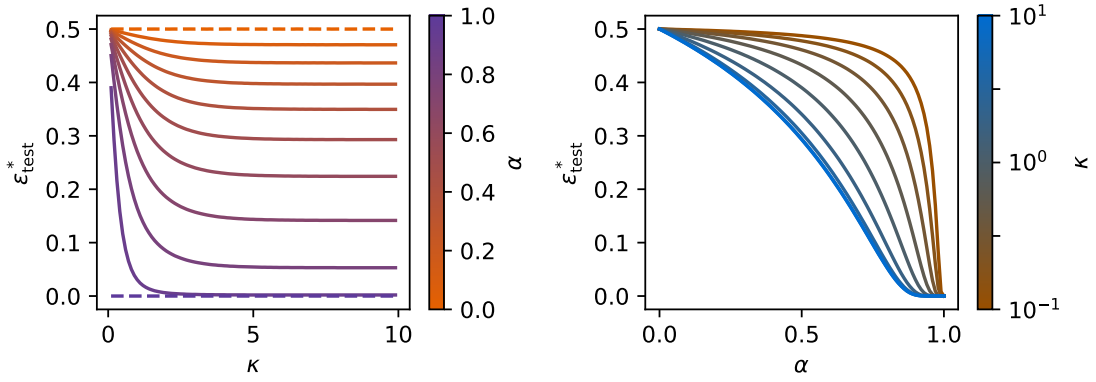


Figure F.9: Error lower bound $\epsilon^*_G(\kappa, \alpha, q = 1/2)$ for finite κ, from numerical minimization of $I(\epsilon|\kappa, q, \alpha)$. (A) $\epsilon^*_G(\kappa)$ for $\alpha = 0, 0.1, \ldots, 0.9, 1.0$ (orange to purple). Dashed lines indicate $\alpha = 0, 1$. At each fixed α, the lower bound on the test error is minimal for the hard likelihood ($\kappa \to \infty$). (B) $\epsilon^*_G(\alpha)$ for $\kappa = 10^{-1}, 10^{-3/4}, \ldots, 10^{3/4}, 10$ (brown to blue). For any value of κ, ϵ^*_G is a decreasing value of α, and the transition from low to high error near $\alpha = 1$ is sharper at smaller κ.

The obtained bound depends on q. Consider in particular the case where q is very small, i.e. the prior is a power law with power α on an input space much larger than the training set. This limit can be analysed for any κ.

For all q and $\alpha < 1$, the derivative of the rate function is

$$\frac{\mathrm{d}I(\epsilon|\kappa, q)}{\mathrm{d}\epsilon} = \alpha \frac{\mathrm{d}H_2(\epsilon)}{\mathrm{d}\epsilon} + \frac{\partial F(\epsilon^*_S|\epsilon, \kappa, q)}{\partial \epsilon} + \underbrace{\frac{\partial F(\epsilon^*_S|\epsilon, \kappa, q)}{\partial \epsilon_S}}_{=0} \frac{\partial \epsilon^*_S}{\partial \epsilon}. \tag{F.52}$$

F can be expanded in series as $q \to 0$, giving

$$F(\epsilon_S|\epsilon, \kappa, q) = -H(\epsilon) + (-\kappa\epsilon_S \log_2 e - H_2(\epsilon_S) - H(\epsilon) + (\epsilon - \epsilon_S)H_2'(\epsilon))q + O(q^2) \tag{F.53}$$

and thus

$$\frac{\mathrm{d}I(\epsilon|\kappa, q)}{\mathrm{d}\epsilon} = (\alpha - 1)\frac{\mathrm{d}H_2(\epsilon)}{\mathrm{d}\epsilon} + (\epsilon - \epsilon^*_S(\epsilon))H_2''(\epsilon)q + O(q^2). \tag{F.54}$$

$(\epsilon - \epsilon_S)H_2''(\epsilon) < \epsilon H_2''(\epsilon)$, which has a finite limit as $\epsilon \to 0$. Thus, as $q \to 0$, if $\alpha < 1$, this derivative can only be zero if the derivative of $H_2(\epsilon) \to 0$.

The derivative of $H_2(\epsilon)$ is only zero at $\epsilon = 1/2$, and thus

$$\lim_{q \to 0} \epsilon^* = \begin{cases} \frac{1}{2}, & \alpha < 1 \\ 0, & \alpha \geq 1. \end{cases} \tag{F.55}$$

Finally, can we say anything about a prior which is not a pure power law? In this case, we also expect to be able to guarantee a nonzero generalisation error in any case where the prior decays "more slowly" than the Zipfian $1/\operatorname{rank} f$.

To formalize this idea, suppose that $P_{r(\epsilon_0)}/P_1 = 1/r^\alpha$ for some *particular* ϵ_0. Fix the unnormalized prior at $r = 1$ to be 1. Then $I(0) = 0$, so the error is nonzero if I is negative for any finite ϵ. Since we have assumed $P_{r(\epsilon_0)}/P_1 = 1/r^\alpha$, $I(\epsilon_0)$ is given by its value for a pure power-law prior, $I(\epsilon_0|\alpha, q)$. Thus, the error is guaranteed to be nonzero as long as $I(\epsilon_0|\alpha, q)$, as in Eq. (F.34), is negative. This is in particular clearly the case if $\epsilon_0 \leq \epsilon^*$, although a stronger bound exists in general.

Note that, for this bound to amount to anything, we need to have $\epsilon_0 > 0$ for large m. This requires our sub-zipfian scaling to hold over a wide range of ranks, since $r(\epsilon_0)$ is exponential in m.

F.6 A super-Zipfian prior cannot learn very many functions

F.6.1 Summary of results

We have shown that a sub-Zipfian prior does not have a strong enough inductive bias to achieve good learning performance, even on a single target function.

An unnecessarily strong inductive bias, however, should also be bad: it may be able to achieve good generalisation for a single target function, but if the bias is already strong enough to allow generalisation, making it stronger only limits the number of functions that can be learned.

Thus, we want to show that $\alpha > 1$ produces some unecessary underfitting, i.e., guarantees nonzero training error ϵ_S.

In fact, for a soft likelihood (κ finite), we do not ever expect zero training error. This likelihood comes from a model where data are assumed to be mislabelled with probability $1/(1 + \exp(\kappa)$, and it is straightforward to show that a uniform prior $P(f) = 1/2^m$ gives $\epsilon_S = 1/(1 + \exp(\kappa))$.

Thus, ideally, we would be able to show that, for almost all target functions, $\epsilon_S > 1/(1+\exp(\kappa))$ for a superzipfian prior (and perhaps $\epsilon_S = 1/(1+\exp(\kappa))$ for a subzipfian prior).

Appendix F.6.3 produces such a result, but under much stronger hypotheses—for a random *ordering of the prior*, it shows that

$$\epsilon_S = \frac{1}{1+e^{\kappa}}, \quad \alpha \leq 1 \tag{F.56}$$

$$\epsilon_S > \frac{1}{1+e^{\kappa}}, \quad \alpha > 1. \tag{F.57}$$

We thus might *conjecture* that for random (i.e., almost all) target functions, the above errors hold for a non-random ordering of the prior.

What we have obtained rigorously in Appendix F.6.2, however, is somewhat weaker. Pick an ϵ', and consider all the possible target functions which differ by ϵ' from the most likely function. We obtain an implicit equation for "$\epsilon_S[\min](\kappa, \alpha, \epsilon')$". For any $\epsilon_S \in (0, \epsilon_S[\min])$, almost all the targets which differ from the most likely function by ϵ' have training error at least ϵ.

$\epsilon_S[\min]$ is given by

$$\alpha \left(H(\epsilon') - H(\epsilon_S[\min])\right) = \epsilon' \kappa \log_2 e, \tag{F.58}$$

This is weaker than what the random-prior result suggests should be possible in two ways:

1. Although we find exponentially many functions with nonzero training error, this isn't actually almost all functions or even a majority of functions. (We only consider functions where you get error ϵ_S because you choose the most likely function instead. To get almost all targets period, you would presumably have to consider the possibility that you get confused not by picking the most likely function, but by picking a different function.)

2. $\epsilon_S[\min]$ is not always larger than the uniform-prior bound $1/(1+\exp(\kappa))$. What we can actually prove is that, for large κ, $\epsilon_S[\min]$ becomes larger than $1/(1+\exp(\kappa))$.

We might, however, say that this is good enough—you get higher training error than expected in precisely the limit where you are trying to get a small training error.

F.6.2 Rigorous results

Consider the $2^{m_{\text{train}}}$ possible functions on the training set. Let $f^{(r)}$ denote the function at rank r in the prior, so that $f^{(1)}$ is the most likely function. For any function $f^{(r)}$, we call the set of functions which differ from it on exactly $m\epsilon$ elements its ϵ–sphere, and the set of functions which differ from it on less than ϵ elements its (open) ϵ–ball. Notice that if $f^{(\mu)}$ is in the ϵ–sphere (ball) of $f^{(\mu)}$, then $f^{(\mu)}$ is in the ϵ–sphere (ball) of $f^{(\nu)}$. Recall that, for the soft likelihood with strength κ, we "expect" to have training error $\epsilon_0(\kappa) = 1/(1+e^{\kappa})$. In this section, we establish the following theorem:

Theorem 9. *Consider Bayesian inference using a power-law prior with exponent $\alpha > 1$, soft likelihood with strength κ, and training examples corresponding to some "target function". Let $\epsilon_{\min}(\alpha, \kappa, \epsilon')$ be defined as the solution to $S_2(\epsilon_{\min}) = S_2(\epsilon') - \frac{\kappa}{\alpha}\epsilon'$. Then for any $\epsilon < \epsilon_{\min}(\alpha, \kappa, \epsilon')$, as $m \to \infty$, all but an exponentially (in m) small fraction of the possible target functions in the ϵ'–sphere of $f^{(1)}$ will have training error at least ϵ. Further, for any $\alpha > 1$, there exists $\kappa_0(\alpha)$ such that, for $\kappa > \kappa_0(\alpha)$, a range of ϵ' have $\epsilon_{\min}(\alpha, \kappa, \epsilon') > \epsilon_0(\kappa)$. More precisely, for $\kappa > \kappa_0(\alpha)$, there exist $\epsilon'_1(\alpha, \kappa), \epsilon'_2(\alpha, \kappa)$ such that, for any $\epsilon' \in (\epsilon'_1(\alpha, \kappa), \epsilon'_2(\alpha, \kappa))$, we have $\epsilon_{\min}(\alpha, \kappa, \epsilon') > \epsilon_0(\kappa)$. For large κ, $\epsilon'_1 \lesssim \frac{\alpha}{\alpha-1}\epsilon_0(\kappa)$, and $\epsilon'_2 \gtrsim \min((1+\sqrt{2})e^{-\kappa/\alpha}, \frac{1}{2})$.*

Thus, for a superzipfian prior, we tend to "underfit" in the following sense. If we try to make the training error small by making κ large, then we can always find exponentially many functions that have a larger training error than they "should" for an expressive prior with this likelihood. This problem becomes more and more severe as the prior becomes steeper: for large α, we underfit functions with distance ϵ' barely any larger than $\epsilon_0(\kappa)$, and extending to $\epsilon' \gg \epsilon_0(\kappa)$.

Proof. For any function, its ϵ–sphere contains $\omega(\epsilon) \equiv \binom{m_{\text{train}}}{m_{\text{train}}\epsilon}$ functions, and its ϵ–ball contains $\Omega(\epsilon) \equiv \sum_{\epsilon'<\epsilon} \omega(\epsilon')$ functions. As $m \to \infty$, the two grow at the same rate (in the large-deviation sense); for any $\epsilon \leq 1/2$ we have

$$\frac{1}{\sqrt{8\epsilon(1-\epsilon)}} e^{mS_2(\epsilon)} \leq \omega(\epsilon) \leq \Omega\left(\epsilon + \frac{1}{m}\right) \leq e^{mS_2\left(\epsilon+\frac{1}{m}\right)}. \tag{F.59}$$

(For $\epsilon > 1/2$ we instead have $\Omega(\epsilon) \asymp 2^m = e^{mS_2(\frac{1}{2})}$, while still $\omega(\epsilon) \asymp e^{mS_2(\epsilon)}$.)

For succinctness, when we refer to "almost all" elements of an exponentially large set, $|S| \asymp 2^{\beta m}$, we will mean all but a number of elements which may even grow exponentially, but with a smaller rate.

Now, we will consider the set F of possible target functions in the ϵ'–ball of $f^{(1)}$, and try to show that almost all of them have training error at least ϵ, for some $\epsilon \leq 1/2$.

For each function in the prior, at most $\Omega(\epsilon)$ functions in F can be in its ϵ–ball. Thus, all the functions of rank up to r include at most $r\Omega(\epsilon)$ functions in their ϵ–balls. Thus, for any $\delta_1 > 0$, almost all of the functions in F aren't in the ϵ–balls of any function with rank less than $r_1 = \omega(\epsilon' - \delta_1)/\Omega(\epsilon)$.

We define the unnormalized posterior $\tilde{Q}_\mu = P_\mu L_\mu$ of a function. For a given target function, let $\tilde{Q}_{\text{true}}$ be the sum of $\tilde{Q}_\mu$ for all functions in its ϵ–ball and $\tilde{Q}_{\text{false}}$ be the sum of $\tilde{Q}_\mu$ for all other functions. If for any function $\tilde{Q}_{\text{false}} \gg \tilde{Q}_{\text{true}}$, then this function will have training error at least ϵ; the normalization of the posterior does not affect this comparison. $\tilde{Q}_{\text{false}}$ is bounded below by the weight on $f^{(1)}$, i.e.,

$$\tilde{Q}_{\text{false}} > e^{-\kappa m_{\text{train}}\epsilon'}. \tag{F.60}$$

Recall that almost all of the functions in F are not in the ϵ–ball of any function with rank less than r_1. Further, each of the functions with rank greater than r_1 can only be in the ϵ–ball of at most $\Omega(\epsilon)$ functions in F. Thus, the *sum* of $\tilde{Q}_{\text{true}}$ for all functions in F is no greater than $\Omega(\epsilon) \sum_{\mu=r_1}^{\infty} P_\mu \sim \Omega(\epsilon) r_1^{1-\alpha}$. Since $\tilde{Q}_{\text{true}}$ cannot be negative, this means that, for any $\delta_2 > 0$, almost all functions in F must have

$$\tilde{Q}_{\text{true}} \leq \frac{e^{m_{\text{train}}\delta_2}\Omega(\epsilon)}{\omega(\epsilon')} \sum_{\mu=r_1}^{\infty} P_\mu \asymp \frac{e^{m_{\text{train}}\delta_2}\Omega(\epsilon)^\alpha}{\omega(\epsilon')\omega(\epsilon'-\delta_1)^{\alpha-1}}. \tag{F.61}$$

If $\tilde{Q}_{\text{true}}$ decays faster as $m \to \infty$ than $\tilde{Q}_{\text{false}}$ for almost all functions under consideration, then for almost all of these functions we will have training error at least ϵ. This condition is equivalent to

$$\lim_{m_{\text{train}} \to \infty} \left(-\delta_2 - \frac{\alpha}{m_{\text{train}}} \log \Omega(\epsilon) + \frac{1}{m_{\text{train}}} \log \omega(\epsilon') + (\alpha - 1) \frac{1}{m_{\text{train}}} \log \omega(\epsilon' - \delta_1) - \kappa \epsilon' \right) > 0, \tag{F.62}$$

$$-\delta_2 - \alpha S_2(\epsilon) + S_2(\epsilon') + (\alpha - 1) S_2(\epsilon' - \delta_1) - \kappa \epsilon' > 0. \tag{F.63}$$

Recall that δ_1, δ_2 must be arbitrary positive constants. Note that, if we were to instead to take $\delta_1, \delta_2 = 0$, then the left-hand side would be positive for $0 < \epsilon < \epsilon_{\min}$, exactly zero for $\epsilon = \epsilon_{\min}(\alpha, \kappa, \epsilon')$, and negative for $\epsilon > \epsilon_{\min}$. By continuity, then, for any $0 < \epsilon < \epsilon_{\min}$, we can choose δ_1, δ_2 positive, but small enough that we still satisfy the above inequality.

Now, we show the second half of the theorem. Consider the function $\varphi_k(x) = S_2(x) - kx$. φ_k is convex, and positive over some interval $(0, y)$. $\varphi_k(x)$ attains its maximum at $x_0(k) = 1/(1 + e^k)$, and $\varphi_k^* \equiv \varphi_k(x_0) = \log\left(1 + e^{-k}\right)$. The "expected" training error for likelihood strength κ is $\epsilon_0 = x_0(\kappa)$. For $k > 1$ we have the bounds $e^{-k} - \frac{1}{2}e^{-2k} \leq \varphi_k(x_0) \leq e^{-k}$ and $(k+1)e^{-k} - (k + \frac{1}{2})e^{-2k} \leq S_2(x_0) \leq (k+1)e^{-k}$, which become tight for large k.

We thus have $\max_{\epsilon'}(S_2(\epsilon') - (\kappa/\alpha)\epsilon') \geq e^{-\kappa/\alpha} - O(e^{-2\kappa/\alpha})$ and $S_2(\epsilon_0(\kappa)) \leq (\kappa + 1)e^{-\kappa}$. Thus, for any $\alpha > 1$ and large enough κ, $\max_{\epsilon'}(S_2(\epsilon') - (\kappa/\alpha)\epsilon') > S_2(\epsilon_0(\kappa))$. By continuity this implies that, for any $\alpha > 1$ and large enough κ, there exists $\epsilon_1'(\alpha, \kappa), \epsilon_2'(\alpha, \kappa)$ such that, for any $\epsilon' \in (\epsilon_1', \epsilon_2')$, $\epsilon_{\min}(\alpha, \kappa, \epsilon') > \epsilon_0(\kappa)$.

Using the exact formulas for f^* and $S_2(x_0)$ can give an implicit equation for $\kappa_0(\alpha)$ From this equation, we see that $\kappa_0(\alpha)$ diverges as $\alpha/(\alpha - 1)\log(\alpha/(\alpha - 1))$ as $\alpha \to 1$, and $\kappa_0(\alpha) \sim 4/\alpha$ for large α.

To establish the values of ϵ_1', ϵ_2', we begin by bounding the value of $\varphi_k(x)$. When φ_k is increasing, a lower bound on φ_k gives an upper bound on its inverse. When φ_k is decreasing, a lower bound on φ_k gives a lower bound on its inverse. Thus, lower bounds on $\varphi_{\kappa/\alpha}(\epsilon')$ allow for both an upper bound on ϵ_1' and a lower bound on ϵ_2'.

We find that different lower bounds are useful in each of these cases. We require two elementary bounds:

$$\varphi_k(x) \geq -x \log x - kx \tag{F.64}$$

$$\varphi_k(x) \geq \varphi_k^* - \frac{(x - x_0(k))^2}{2 \min(x(1-x), x_0(k)(1-x_0(k))}. \tag{F.65}$$

The first inequality follows from $-(1-x)\log(1-x) > 0$ for $0 < x < 1$. The second can be derived by applying Taylor's remainder theorem to $\varphi_k(x)$ around the point $x_0(k)$ and considering the minimum value of φ_k'' on the interval between x_0 and x.

The first bound, combined with our upper bound on $S_2(\epsilon_0(\kappa))$, yields

$$\epsilon_1' \leq -\frac{(\kappa+1)e^{-\kappa}}{W_{-1}(-(\kappa+1)e^{\kappa/\alpha-\kappa})}. \tag{F.66}$$

(where $W_{-1}(x)$ is the lower branch of the product logarithm function). The bound $W_{-1}(x) \leq \log(-x)$ (i.e., $-1/W_{-1}(x) \leq -1/\log(-x)$) then gives

$$\epsilon_1' \leq \frac{\alpha(1+\kappa)e^{-\kappa}}{(\alpha-1)\kappa - \alpha\log(1+\kappa)} \approx \frac{\alpha}{\alpha-1}\epsilon_0(\kappa), \tag{F.67}$$

with the approximation holding for large κ and $\alpha > 1$.

We use the second lower bound on φ_k, again combined with our upper bound on $S_2(\epsilon_0(\kappa))$, to obtain a lower bound on ϵ_2'. We have $\epsilon_2' > x_0(\kappa/\alpha)$; if we further assume $\epsilon_2' \leq \frac{1}{2}$ then the point where the lower bound on $\varphi_{\kappa/\alpha}(\epsilon_2')$ crosses the upper bound on $S_2(\epsilon_0(\kappa))$ becomes the solution to a quadratic equation.

We arrive at the simple expression

$$\epsilon_2' \geq \epsilon_0(\kappa/\alpha) + \sqrt{2[\varphi_{\kappa/\alpha}^* - S_2(\epsilon_0(\kappa))]\epsilon_0(\kappa/\alpha)(1-\epsilon_0(\kappa/\alpha))} \tag{F.68}$$

$$\geq \epsilon_0(\kappa/\alpha) + \sqrt{2\varphi_{\kappa/\alpha}^*\epsilon_0(\kappa/\alpha) + O(e^{-\kappa}) + O(e^{-2\kappa/\alpha})} \tag{F.69}$$

$$\gtrsim (1+\sqrt{2})e^{-\kappa/\alpha}. \tag{F.70}$$

with the final approximation again holding for large κ and $\alpha > 1$.

However, we recall that we needed to assume $\epsilon_2' \leq \frac{1}{2}$ to simplify the equation, so in fact we have shown that

$$\epsilon_2' \gtrsim \min\left((1+\sqrt{2})e^{-\kappa/\alpha}, \frac{1}{2}\right). \tag{F.71}$$

Inspecting plots suggests that the $\frac{1}{2}$ could be improved if desired.

□

F.6.3 Superzipf priors increase the training error for a random function

The goal of this section is to establish that, for a random ordering of the functions in the prior, as $m \to \infty$,

$$\epsilon_S = \frac{1}{1+e^{\kappa}}, \quad \alpha \leq 1 \tag{F.72}$$

$$\epsilon_S > \frac{1}{1+e^{\kappa}}, \quad \alpha > 1. \tag{F.73}$$

Note that this *does not* follow from the worst-case style bound above, which goes to zero for any finite κ as $\alpha \to 1$.

To do the average over a random ordering of the prior, we will need to make use of the replica trick. We introduce the notation $\overline{\cdot}$ for an average over the ordering of the prior. Defining the partition function

$$Z = \sum_{r=1}^{2^m} P(r) e^{-\kappa m \epsilon(r)}, \tag{F.74}$$

we have

$$\overline{\epsilon_S} = -\frac{1}{m} \partial_\kappa \overline{\log Z}. \tag{F.75}$$

As usual, we write

$$\overline{\log Z} = \lim_{n \to 0^+} \partial_n \overline{Z^n}. \tag{F.76}$$

Also as usual, we will (1) evaluate $\overline{Z^n}$ for integer n, and analytically continuing it to non-integer n, and (2) interchanging the limits $m \to \infty$ and $n \to 0$ in order to use the saddle-point approximation at nonzero n.

At a fixed ordering of the functions, the replicated partition function is

$$Z^n = \sum_{r_1=1}^{2^m} \sum_{r_2=1}^{2^m} \cdots \sum_{r_n=1}^{2^m} P(r_1) \cdots P(r_n) e^{-\kappa m \sum_{i=1}^{n} \epsilon(r_i)}. \tag{F.77}$$

The disorder averages depends how many of the n replicas have the same function r_i. We adopt the standard variational ansatz used in the solution of the random energy model, where the replicas form k groups, each of which has $q \equiv n/k$ members that have the exact same function. (This is an ansatz that allows

for one-step replica symmetry breaking.) The contribution of this state to the replicated partition function is

$$Z_k^n = \sum_{r_1=1}^{2^m} \cdots \sum_{r_k=1}^{2^m} P(r_1)^q \cdots P(r_k)^q e^{-\kappa q m \epsilon(r_i)} \cdots e^{-\kappa q m \epsilon(r_k)}. \tag{F.78}$$

Since we will always be working in the limit $k \ll 2^m$, we can approximate $\epsilon(r_1), \ldots, \epsilon(r_k)$ as uncorrelated. (If $k \sim 2^m$, they would be correlated because the errors of each state are drawn without replacement.) This allows us to perform the disorder average, using

$$\overline{e^{-q\kappa m \epsilon(r_i)}} = \sum_{j=1}^{m} \binom{m}{j} e^{-q\kappa j} = \left(\frac{1+e^{-q\kappa}}{2}\right)^m, \tag{F.79}$$

giving

$$\overline{Z_q^n} = \sum_{r_1=1}^{2^m} \cdots \sum_{r_k=1}^{2^m} P(r_1)^q \cdots P(r_j)^q \left(\frac{1+e^{-q\kappa}}{2}\right)^{mk} = \left(\sum_r P(r)^q\right)^{n/q} \left(\frac{1+e^{-\kappa q}}{2}\right)^{mn/q}. \tag{F.80}$$

We next simplify the free energy functional $f_n(k) = \log \overline{Z_q^n}$ in the limit $m \to \infty$. Here we make some approximations that come from $n \ll 1$, which also gives $0 < q < 1$ as in the REM solution. Dropping subextensive terms, the free energy functional simplifies in this limit to

$$-f_n(q) = \begin{cases} \left[-\log 2 + \frac{1}{q}\log(1+e^{-q\kappa})\right] nm, & \alpha \le 1 \\ \left[-\min(\alpha, \frac{1}{q})\log 2 + \frac{1}{q}\log(1+e^{-q\kappa})\right] nm, & \alpha > 1. \end{cases} \tag{F.81}$$

As usual, when we flip q to be less than 1 rather than greater than 1, we need to maximize this free energy rather than minimizing it. For $\alpha \le 1$ the maximum is at the boundary point $q = 1$. For $\alpha > 1$, the maximum is at $q = 1/\alpha$. This straightforwardly gives:

$$\overline{\log Z} = \begin{cases} \left[\log(1+e^{-\kappa}) - \log 2\right] m, & \alpha \le 1 \\ \alpha\left[\log\left(1+e^{-\kappa/\alpha}\right) - \log 2\right] m, & \alpha > 1, \end{cases} \tag{F.82}$$

and thus

$$\overline{\epsilon_S} = \begin{cases} \frac{1}{1+e^{\kappa}}, & \alpha \le 1 \\ \frac{1}{1+e^{\kappa/\alpha}}, & \alpha > 1. \end{cases} \tag{F.83}$$

F.7 The PAC-Bayesian bound

When the prior is known over the full space of m inputs, we established *lower bounds* on generalisation error, which indicates that for $\alpha > 1$ learning is poor. Now we will assume only that the prior is known over the m_{train} training data, and investigate the behaviour of conventional PAC–Bayesian bounds, which establish *upper bounds* on generalisation error.

We consider a PAC–Bayesian bound due to Valle-Pérez [63], which is formulated in terms of the prior probability of a function as defined solely on the m_{train} training data. We will show that this bound is optimal for $\alpha = 1$ in the following sense:

- For $\alpha < 1$, the PAC–Bayesian bound remains nonzero as $m_{\text{train}} \to \infty$ regardless of the training data: it is impossible to guarantee an arbitrarily small test error.
- For $\alpha \geq 1$, as $m_{\text{train}} \to \infty$, it is possible to guarantee test error below ϵ, for any ϵ, for target functions up to rank r_0 on the training data. Super-Zipfian priors, however, dramatically reduce r_0: for $\epsilon \ll 1$, we find that $(1/m_{\text{train}}) \log r_0 = \epsilon/\alpha$.

A function f defined by its output on the training set S (using S_X and S_Y to denote the sets of inputs and labels repsectively as in Appendix F.3) of m_{train} training examples has some probability $P(f|S)$; this probability is related to the prior P over functions defined on the full input space m by marginalizing over the classification of examples not in the training set (this is the same as a measurement of probability made while ignoring the output of the function on unobserved inputs, as in our numerical studies). The PAC–Bayesian bound of Valle-Pérez [63] then states that, for a function f sampled from the posterior with hard likelihood, with probability $1 - \delta$ over the choice of training set and $1 - \gamma$ over the sampling from the posterior,

$$-\ln(1 - \epsilon(f)) \leq \frac{-\ln P(f^*|S_X) + \ln\left(\frac{m_{\text{train}}}{\delta\gamma}\right)}{m_{\text{train}} - 1}, \tag{F.84}$$

where f^* is the function on the m_{train} inputs consistent with the training data labels S_Y. Let $r^* = \operatorname{rank} f^*$ and $r_\nu = \operatorname{rank} f^{(\nu)}$.

Assuming $P(f^{(\nu)}|S_X) = A(m_{\text{train}})/r_\nu^\alpha$, this bound becomes

- $\alpha < 1$:

$$-\ln P(f^*; S) = \alpha \ln r^* + \ln\left(2^{m_{\text{train}}(1-\alpha)} - 1\right) - \ln(1-\alpha) \tag{F.85}$$

- $\alpha = 1$:

$$-\ln P(f^*; S) = \ln r^* + \ln m_{\text{train}} + \ln\ln 2 \tag{F.86}$$

- $\alpha > 1$:

$$-\ln P(f^*; S) = \alpha \ln r^* + \ln\left(1 - 2^{m_{\text{train}}(1-\alpha)}\right) - \ln(\alpha - 1). \tag{F.87}$$

F.8 Mechanistic origins of Zipf's law in the Kernel limit

In the limit of infinite width, neural networks are equivalent to kernel machines. Thus, if Zipf's law is a generic phenomenon for reasonable neural network priors, it should also be seen in kernel machine priors.

Here, we will show how Zipf's law emerges in kernel priors in various cases and simplifying limits. In Appendix F.8.1 we introduce an argument from [109] that explains how latent variables can produce Zipf's law. In Appendix F.8.2 we apply this argument to kernels, showing that the projections of a function onto the "learnable" (i.e., high-eigenvalue) eigenmodes of a kernel are the relevant latent variables. In Appendix F.8.3 we then work this connection between Zipf's law and an underlying latent variable in more detail for the equicorrelated kernel. We provide experimental results in Appendix F.8.4 and discuss the connections between the equicorrelated kernel and infinite width and depth chaotic networks in Appendices F.8.5 and F.8.6.

F.8.1 Zipf's law from low-dimensional latent variables

There are many models which can produce power-law distributions $P_\nu = 1/r_\nu^\alpha$, however, they often need to be fine-tuned to give a power of 1. The main known mechanism which *robustly* produces Zipf's law is when high-dimensional observations (such as the m-dimensional binary string $f^{(\nu)}$) are driven by a much lower-dimensional "latent variable" [108, 109]. The simplest example, shown in [108], is uncorrelated Ising spins driven by a fluctuating external magnetic field (the latent variable).

We define the "energy" of a given state ν as

$$E_\nu = -\log P_\nu. \tag{F.88}$$

In terms of E, there are two related characterizations of Zipf's law. One may show [261] that Zipf's law requires

$$S(E) = E + \text{const.} \tag{F.89}$$

where $S(E)$ is the entropy of states with a given E, i.e. the log of their density, or equivalently [109]

$$\log r_\nu = E_\nu + \text{const.} \tag{F.90}$$

In either case, if $E = O(m)$, then as m becomes large it suffices to show that variations in the correction term are $o(m)$, rather than it being strictly constant.

From equation Eq. (F.89) it is intuitive how latent variables can lead to Zipf's law. Suppose different functions f are distinguished by some "order parameter" $\mathbf{x}$, which is coupled to a fluctuating latent variable $\boldsymbol{z}$ (i.e., we first sample $\boldsymbol{z}$, and then $\boldsymbol{z}$ influences f by making certain values of $\mathbf{x}$ much more likely). Suppose all states with a given value $\mathbf{x}$ are equally likely. Then for any function f with a given value of $\mathbf{x}$, we have

$$P(f) = \frac{P(\mathbf{x})}{e^{S(\mathbf{x})}} \tag{F.91}$$

$$E(f) = S(\mathbf{x}) - \log P(\mathbf{x}). \tag{F.92}$$

Then, as long as (1) the dimension of $\mathbf{x}$ is subextensive so that $S(E(\mathbf{x})) = S(\mathbf{x}) + o(m)$, (2) $P(\mathbf{x})$ is roughly flat (i.e. variations in $\log P(\mathbf{x})$ are $o(m)$), and (3) distinct values of $\mathbf{x}$ actually have distinct entropies as $m \to \infty$, we will have Zipf's law as $m \to \infty$. In most scenarios where large-dimensional data are controlled by some low-dimensional latent variable $\boldsymbol{z}$, we expect the distribution of an order parameter $\mathbf{x}$ that couples to $\boldsymbol{z}$ to be closely related to the distribution of $\boldsymbol{z}$ itself. For example, in the Ising case, $\mathbf{x}$ is the magnetization and $\boldsymbol{z}$ is the magnetic field, and unless we sit at a critical point, the distribution $P(\mathbf{x}|\boldsymbol{z})$ becomes exponentially narrow as $m \to \infty$, meaning that a broad distribution $P(\boldsymbol{z})$ guarantees a broad distribution $P(\mathbf{x})$ as $m \to \infty$.

Note that, in principle, this argument only guarantees Zipf's law for large ranks, since it takes $m \to \infty$ at fixed $\boldsymbol{z}$.

In particular, note that a "complexity bias" is one possible source of Zipf's law by the above argument [5]. If the probability of a function f were determined solely by its Kolmogorov complexity K, then since $S(K) \sim K$ as $m \to \infty$, Zipf's law will emerge if the distribution $P(K)$ is roughly flat as $m \to \infty$.

In cases where it cannot be assumed that $P(f)$ is identical for functions with the same value of the order parameter, or the order parameter cannot be clearly mapped to the latent variable $\boldsymbol{z}$, more general versions of the argument can be constructed. When the latent variable $\boldsymbol{z}$ is finite-dimensional and $\log P(f|\boldsymbol{z}) = m \sum m_i(f) z_i$, Ref. [108] obtains $E = S + o(m)$ by saddle-point approximations of E and S. [109] instead start with Eq. (F.90). They show that

$$\log r_\nu = E_\nu + \log \int_{-\infty}^{E_\nu} \mathrm{d}E'\, P(E') e^{E'-E_\nu} \equiv E_\nu + \log P_S(E_\nu), \tag{F.93}$$

and thus Zipf's law emerges if $P_S(E)$ is roughly constant over a wide range of ranks, which requires a broad distribution of energies E_ν.

They further show that, for m-dimensional states with large m, a "broad enough" distribution of energies is achieved if $\mathrm{Var}[E] = O(m^2)$. This is because, under this condition, an $O(m)$ change in E only produces an $O(1)$ change in $\log P_S(E)$ [109].

This large variance in E can be achieved by identifying a latent variable $\boldsymbol{z}$ that produces $O(m)$ changes in E. This is because the law of total variance implies that

$$\mathrm{Var}_\nu[E(\mathbf{f})] = \mathrm{Var}_{\boldsymbol{z}}[\mathbb{E}_{\mathbf{f}|\boldsymbol{z}}[E(\mathbf{f})]] + \mathbb{E}_{\boldsymbol{z}}[\mathrm{Var}_{\mathbf{f}|\boldsymbol{z}}[E(\mathbf{f})]] \geq \mathrm{Var}_{\boldsymbol{z}}[\mathbb{E}_{\mathbf{f}|\boldsymbol{z}}[E(\mathbf{f})]], \tag{F.94}$$

and thus the if $\boldsymbol{z}$ produces $O(m)$ changes in E, and the distribution of $\boldsymbol{z}$ does not become narrow as $m \to \infty$, we will have $\mathrm{Var}[E] = O(m^2)$.

If $\dim \boldsymbol{z} = o(m)$ and the components f_i are conditionally independent for each $\boldsymbol{z}$, then this may be further approximated, following Ref. [109]. To estimate the lower bound on the right, we note that

$$\mathbb{E}_{\mathbf{f}|\boldsymbol{z}}[E(\mathbf{f})] = -\sum_{\mathbf{f}} P(\mathbf{f}|\boldsymbol{z}) \log P(\mathbf{f}|\boldsymbol{z}) + \sum_{\mathbf{f}} P(\mathbf{f}|\boldsymbol{z}) \log \frac{P(\mathbf{f}|\boldsymbol{z})}{P(\mathbf{f})}. \tag{F.95}$$

The second term is positive definite, and its expectation is the mutual information between $\mathbf{f}$ and $\boldsymbol{z}$, which is bounded above by the entropy of the latent variable and is thus $O(\dim(\mathbf{z})) = o(m)$. The first term is $O(m)$. Thus, the second term may be neglected. This gives us

$$\mathrm{Var}_\nu[E(\mathbf{f})] \gtrsim \mathrm{Var}_{\boldsymbol{z}}\left[\sum_{\mathbf{f}} P(\mathbf{f}|\boldsymbol{z}) \log P(\mathbf{f}|\boldsymbol{z})\right] \equiv \mathrm{Var}_{\boldsymbol{z}}\left[S(\mathbf{f}|\boldsymbol{z})\right]. \tag{F.96}$$

Then, if f_i are conditionally independent for each $\boldsymbol{z}$, we may us $S(\mathbf{f}|\mathbf{z}) = \sum_i S(f_i|\mathbf{z})$ to write

$$\mathrm{Var}_{\mathbf{h}}[E(\mathbf{f})] \gtrsim \sum_{ij} \mathrm{Cov}[S(f_i|\mathbf{h}), S(f_j|\mathbf{h})]. \tag{F.97}$$

F.8.2 Zipf's law from kernels

In the context of kernel regression, other work suggests that a "strong enough" inductive bias to enable generalisation comes from sufficiently fast decay of the kernel eigenvalues, and that generalisation then occurs for target functions which are well-aligned with this inductive bias [100, 112, 96, 98]. In this picture, at a given sample size m, a certain number of high-eigenvalue kernel eigenfunctions are "learnable" (if they are present in the target function), and the remainder are not, and are "benignly overfit" [113] to the training data.

A given kernel, together with a distribution of input data, is characterized by a sequence of kernel eigenvalues Λ_μ and eigenfunctions Φ_μ. The probabilities of functions $P(f)$ on a particular set of inputs, however, are determined instead by a particular finite kernel matrix K. As $m \to \infty$, the μ-th eigenvalue and eigenvector of K converge to $m\Lambda_\mu$ and the associated eigenfunction. Our calculations will reduce to sums of the form $\sum_{\mu=\mu_1}^{\mu_2} \lambda_\mu \approx m \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu$. Using results from [262], one may show (Appendix F.10), for a kernel spectrum decaying like $1/j^\ell$, the relative error in this approximation goes to zero as long as $\mu_1 \sim m^a$ with $a < a_{\max} \equiv \min(\frac{1}{2+\ell/2}, \frac{1}{4+\ell} + \frac{1}{2(\ell-1)})$, which will suffice for our purposes.

To produce a technical simplification, we consider the kernel limit of neural networks which classify *probabilistically*, using the softmax of the final output, instead of deterministically using the sign of the classification as we have studied numerically. With this simplification, we are able to show that Zipf's law emerges for kernels whose eigenvalues λ_k decay like $\lambda_k^{-\ell}$ with $\ell > 1$,under the assumption that the high-rank eigenvectors are random. Since kernel regression is known to generalize for $\ell > 1$, this suggests that kernels which allow for learning will generally give Zipf's law.

Given the Gram (kernel) matrix K for a particular set of m inputs, the probability of a given function f is

$$P(\mathbf{f}) = \frac{1}{\sqrt{2\pi|K|}} \int \mathrm{d}\mathbf{y}\, e^{-\beta \frac{1}{2} \mathbf{y}^T K^{-1} \mathbf{y}} \prod_{i=1}^{n} \frac{e^{\beta y_i f_i}}{2\cosh \beta y_i} \tag{F.98}$$

The kernel may be diagonalized into eigenmodes $\mathbf{v}_\mu$, with eigenvalues λ_μ. As $m \to \infty$, each eigenvalue λ_μ converges to $m\Lambda_\mu$, where Λ_μ is are often referred to as the spectrum of the kernel. Define $\mathbf{h}^{(\mu)}$ as the sampled projection onto the μth eigenvector, $\mathbf{y} \cdot \mathbf{v}_\mu$, and define $w_\mu(f) = \mathbf{v}_\mu \cdot \mathbf{f}$.

Further, let us split the eigenmodes of the kernel into m^a "learnable modes" $\mathbf{h}$ and $m - m^a \approx m$ "unlearnable modes" $\mathbf{h}'$, with $a < 1$. This splitting of the modes is, at this time, an artificial construction, for use in proving Zipf's law holds: we are free to set the value of a, and will do so later. We know this step is required because we plan to use Eq. (F.97), which requires that the latent variable has $o(m)$ dimension.

The probability of a given function f is

$$P(\mathbf{f}) = \int \mathrm{d}\mathbf{h}\,\mathrm{d}\mathbf{h}'\,P(\mathbf{h})P(\mathbf{h}')\frac{e^{-m\log 2-\sum h^{(\mu)}w_\mu(\mathbf{f})-\sum h'^{(\mu)}w'_\mu(\mathbf{f})}}{\prod_{i=1}^m \cosh\Big(\beta(\sum h^{(\mu)}v_{\mu,i}+\sum h'^{(\mu)}v'_{\mu,i})\Big)}. \tag{F.99}$$

We aim to show that, for some values of a and ℓ, (1) the unlearnable modes make an $o(m)$ contribution to the energy of a given function, and then that (2) when only the learnable modes are considered, we can carry through an argument like that in [109] to demonstrate that Zipf's law will be observed.

Using the aforementioned error bound, we may replace sums of the form $\sum_\mu \lambda'_\mu$ with their approximation in terms of the true kernel eigenvalues so long as we choose $a < (\ell-1)/[\ell(4+\ell)]$

First, we seek a bound of the form

$$e^{-b(m)|\mathbf{h}'|} \le e^{-\sum h'^{(\mu)}w'_\mu(\mathbf{f})} \le e^{b(m)|\mathbf{h}'|} \tag{F.100}$$

for some $b(m)$. This will allow us to bound the contribution of the unlearnable modes to the numerator, and thus find conditions which allow us to neglect them.

Since $\mathbf{f}$ is a vector of norm $\sqrt{m}$, Eq. (F.100) is a hard bound if we take $b(m) = \sqrt{m}$, while it holds with probability 1 as $m \to \infty$ if we assume that the unlearnable eigenvectors are random and take $b(m) = 1$.

We may thus write

$$\int \mathrm{d}\mathbf{h}\,\mathrm{d}\mathbf{h}'\,\frac{P(\mathbf{h})P(\mathbf{h}')e^{-m\log 2-\sum h^{(\mu)}w_\mu(\mathbf{f})-b(m)|\mathbf{h}'|}}{\prod_{i=1}^m \cosh\Big(\beta(\sum h^{(\mu)}v_{\mu,i}+\sum h'^{(\mu)}v'_{\mu,i})\Big)} \le P(\mathbf{f}) \tag{F.101}$$

$$\le \int \mathrm{d}\mathbf{h}\,\mathrm{d}\mathbf{h}'\,\frac{P(\mathbf{h})P(\mathbf{h}')e^{-m\log 2-\sum h^{(\mu)}w_\mu(\mathbf{f})+b(m)|\mathbf{h}'|}}{\prod_{i=1}^m \cosh\Big(\beta(\sum h^{(\mu)}v_{\mu,i}+\sum h'^{(\mu)}v'_{\mu,i})\Big)}. \tag{F.102}$$

In order to be able to separate out the contribution of the unlearnable modes to $\log P$, we want bounds to the denominator which can be factorized into separate terms involving h and h'. Unfortunately, these bounds are quite weak. Firstly, if we want to stay in the realm of hard, rigorous bounds which assume nothing about the structure of the modes, we can write:

$$\cosh x\, e^{-|y|} \le \cosh(x+y) \le \cosh x\, e^{|y|}. \tag{F.103}$$

This yields the quite anemic bound

$$e^{-m|\mathbf{h}'|}\prod_{i=1}^{m}\cosh\Big(\beta(\sum h^{(\mu)}v_{\mu,i})\Big)\leq\prod_{i=1}^{m}\cosh\Big(\beta(\sum h^{(\mu)}v_{\mu,i}+\sum h'^{(\mu)}v'_{\mu,i})\Big) \tag{F.104}$$

$$\leq e^{m|\mathbf{h}'|}\prod_{i=1}^{m}\cosh\Big(\beta(\sum h^{(\mu)}v_{\mu,i})\Big). \tag{F.105}$$

This is quite a bit worse than the $b(m)=\sqrt{m}$ we can obtain with no assumptions for the numerator.

What about if the unlearnable modes are assumed random? Under this condition, the quantity we are trying to ignore inside the cosh $O(|\mathbf{h}'|/\sqrt{m})$; we will find below that this is $\ll 1$ for any $a>0$ as $m\to\infty$. We then obtain by Taylor expansion (and defining for convenience $u_i=\sum h^{(\mu)}v_{\mu,i}$, u'_i similarly),

$$\log\prod_{i=1}^{m}\cosh(\beta(u_i+u'_i))\approx\sum_{i}^{m}(\log\cosh(\beta u_i)+u'_i\tanh\beta u_i) \tag{F.106}$$

Under the assumption of random unlearnable eigenvectors, for large m, we may treat sums of functions of u_i and u'_i as averages of random variables and treat u'_i as uncorrelated from u_i. The signs of each u'_i are further uncorrelated, and $u'_i=O(|\mathbf{h}'|/\sqrt{m})$, and thus the full term is $O(|h'|)$.

Thus, in the random case, the correction from the denominator is of the same form $e^{\pm k|h'|}$ as the correction from the numerator. We can absorb it into a new constant $c=\beta(b+k)$, and write

$$\log\int d\mathbf{h}'\,P(\mathbf{h}')e^{-c|\mathbf{h}'|}\leq\log P(\mathbf{f})-\log\int d\mathbf{h}\,\frac{P(\mathbf{h})e^{-m\log 2-\sum h^{(\mu)}w_\mu(\mathbf{f})}}{\prod_{i=1}^{m}\cosh(\beta(\sum h^{(\mu)}v_{\mu,i}))}\leq\log\int d\mathbf{h}'\,P(\mathbf{h}')e^{c|\mathbf{h}'|}. \tag{F.107}$$

We want to show that the unlearnable modes make a subextensive contribution to the energy of a function, i.e. that

$$\log P(\mathbf{f})=\log\int d\mathbf{h}\,\frac{P(\mathbf{h})e^{-m\log 2-\sum h^{(\mu)}w_\mu(\mathbf{f})}}{\prod_{i=1}^{m}\cosh(\beta(\sum h^{(\mu)}v_{\mu,i}))}+o(m). \tag{F.108}$$

To do so, we must show that

$$\log\int d\mathbf{h}'\,P(\mathbf{h}')e^{\pm c|\mathbf{h}'|}=o(m), \tag{F.109}$$

For the case with the minus sign, using Jensen's inequality and the fact that $e^{-|x|} \leq 1$, we have

$$0 \geq \log \int \mathrm{d}\mathbf{h}'\, P(\mathbf{h}') e^{-c|\mathbf{h}'|} \geq -c \int \mathrm{d}\mathbf{h}'\, P(\mathbf{h}')\, |\mathbf{h}'|\,. \tag{F.110}$$

Since $\mathbb{E}[|\mathbf{h}|]^2 \leq \mathbb{E}[|\mathbf{h}|^2] = \sum \lambda'_\mu$, we thus have

$$0 \geq \log \int \mathrm{d}\mathbf{h}'\, P(\mathbf{h}') e^{-c|\mathbf{h}'|} \geq -c\sqrt{\sum \lambda'_\mu}. \tag{F.111}$$

For the case with the plus sign, we, integrate the definition of the expectation by parts and use $e^{c|\mathbf{h}'|} \geq 0$ to obtain

$$\int \mathrm{d}\mathbf{h}'\, P(\mathbf{h}') e^{c|\mathbf{h}'|} = \int_0^\infty \mathrm{d}u\, P(e^{c|\mathbf{h}'|} > u) = \int_0^\infty \mathrm{d}r\, P(|\mathbf{h}'| > r) c e^{cr}. \tag{F.112}$$

A short calculation shows that

$$P(|\mathbf{h}'| > r) \leq 2e^{-r^2/\left(2\sum \lambda'_\mu\right)}, \tag{F.113}$$

which yields

$$\int \mathrm{d}\mathbf{h}'\, P(\mathbf{h}') e^{c|\mathbf{h}'|} \leq 4\sqrt{2\pi} c \sqrt{\sum \lambda'_\mu}\, e^{\frac{c^2}{2}\sum \lambda'_\mu}. \tag{F.114}$$

Thus, taking a logarithm, the contribution to the energy is $O(c^2 \sum \lambda'_\mu)$. This is harder to control than the $O(c\sqrt{\sum \lambda'_\nu})$ bound when the contribution is negative, so we consider it here. We have

$$\sum \lambda'_\mu \sim m^{1+(1-\ell)a}. \tag{F.115}$$

Thus, our condition to have the unlearnable modes make an $o(m)$ contribution for random unlearnable eigenvectors is

$$(\ell - 1)\, a > 0. \tag{F.116}$$

We thus require $\ell > 1$, as is required for e.g. generalisation with ridge regression, and further require $a > 0$. (We can see that this also gives $|\mathbf{h}'|/\sqrt{m} \to 0$, as promised to carry out the expansion Eq. (F.106).)

Most functions must have $E = O(m)$ [109], and thus under the above conditions we can neglect the unlearnable modes. Now we do so, and consider the probability

of each function solely using the learnable modes, which are our latent variables as in [109]. If the resulting variance of energy is at least $O(m^2)$, then Zipf's law will be observed.

For this probabilistic softmax classification, the classification of each site is conditionally independent, and since we have chosen $a < 1$, the number of latent variables is sublinear in m, and thus we may use Eq. (F.97), i.e.

$$\mathrm{Var}_{\mathbf{h}}[E(\mathbf{f})] \gtrsim \sum_{ij} \mathrm{Cov}[S(f_i|\mathbf{h}), S(f_j|\mathbf{h})]. \tag{F.117}$$

To establish that Zipf's law holds, we need to show that most terms in this sum are $O(1)$. To do so, recall the variables $u_i = \sum_\mu h_\mu v_{\mu,i}$ are Gaussian, as linear transformations of Gaussian variables. The covariance of the entropy is then a covariance of a particular nonlinear function of u_i. Define $\hat{u}_i = u_i/\sqrt{\mathrm{Var}[u_i]}$, and $\varphi_i(x) \equiv S_2(e^{\beta\sigma_i x}/2\cosh(\beta\sigma_i x))$. To compute this covariance, we may expand φ in the probabilists' Hermite polynomials, $\varphi_i(x) = \sum_{n=0}^\infty s_{i,2n}\,\mathrm{He}_{2n}(x)$. (The expansion only contains even powers because φ_i is even.) We then have

$$\mathrm{Cov}[S(f_i|\mathbf{h}), S(f_j|\mathbf{h})] = \sum_{m,n=0}^\infty s_{i,2m}s_{i,2n}\mathrm{Cov}[\mathrm{He}_{2m}(\hat{u}_i), \mathrm{He}_{2n}(\hat{u}_j)] = \sum_{n=1}^\infty n!\, s_{i,2n}^2\rho^{2n}. \tag{F.118}$$

Each term in this sum is non-negative. Thus, to show that this covariance is $O(1)$ as $m \to \infty$, it suffices to show that $s_{i,2n}$ remains $O(1)$ and that the correlation ρ between u_i and u_j is $O(1)$. Because the Hermite polynomials are not homogeneous, in general $s_{i,2n}$ has a complicated dependence on σ_i. However, one straightforward case in which this is true is if σ_i, σ_j, and $\mathrm{Cov}[u_i, u_j]$ are *all* $O(1)$ as $m \to \infty$ (at least, for a finite fraction of pairs i, j).

The variance obeys

$$\sigma_i^2 = \sum_\mu \lambda_\mu v_{\mu,i}^2. \tag{F.119}$$

For each μ, for large m the eigenvector converges to the associated eigenfunction of the kernel function. Thus, if the kernel is smooth, the learnable eigenvectors cannot localize on a small number of sites, and typical elements must be $O(1/\sqrt{m})$. Since $\lambda_\mu \sim m$, we thus have $\sigma_i^2 = O(1)$ for any normalizable kernel spectrum, for typical i.

Thus we only need to provide sufficient conditions for $O(1)$ covariance (of either sign). We have

$$\mathrm{Cov}[u_i, u_j] = \sum_\mu \lambda_\mu v_{\mu,i} v_{\mu,j}. \tag{F.120}$$

If and only if this quantity is $O(1)$ (of either sign), its square will be $O(1)$. Its square is:

$$(\mathrm{Cov}[u_i, u_j])^2 = \sum_{\mu,\nu} \lambda_\mu \lambda_\nu v_{\mu,i} v_{\mu,j} v_{\nu,i} v_{\nu,j}. \tag{F.121}$$

Heuristically, we treat the signs of $v_{\mu,i}$ and $v_{\nu,j}$ as uncorrelated. Averaging over this sign, we then get

$$(\mathrm{Cov}[u_i, u_j])^2 = \sum_\mu \lambda_\mu^2 v_{\mu,i}^2 v_{\mu,j}^2. \tag{F.122}$$

Again, if the kernel matrix is derived from a normalizable kernel function, each $v_{\mu,i} = O(1/m)$ and each $\lambda_\mu = O(m)$, and the resulting sum is $O(1)$.

Thus, we will see Zipf's law at large ranks as $m \to \infty$. What determines the strength of deviations from Zipf's law at finite m? On one hand, recall that the latent variable argument for Zipf from the learnable modes requires neglecting terms of relative $O(m^{(a-1)})$ in order to obtain Eq. (F.97). Thus, the smaller we may take a, the better controlled these deviations are. On the other hand, recall that neglecting the unlearnable modes requires neglecting terms of relative $O((1-\ell)a)$, and also involved approximations that require $a < a_{\max} \equiv \min(\frac{1}{2+\ell/2}, \frac{1}{4+\ell} + \frac{1}{2(\ell-1)})$.

We cannot obtain rigorous guarantees because it is unknown how exactly errors in each of these approximations result in deviations from Zipf's law. However, note that $(\ell-1)a_{\max}$ grows monotonically with increasing ℓ. Thus, simply taking $a \approx a_{\max}$, both approximations become better controlled as ℓ increases.

Thus, we expect finite-m deviations from Zipf's law to be more pronounced for slower decays of the kernel spectrum.

We note that this result means only one learnable eigenvalue is sufficient for Zipf's law. In Fig. F.10 we show that increasing the number of learnable eigenmodes leads to a less blocky Zipf's law. We generate a kernel by generating $\dim(h)$ large eigenvalues and $m - \dim(h)$ small eigenvalues (small eigenvalues of size 0.1,

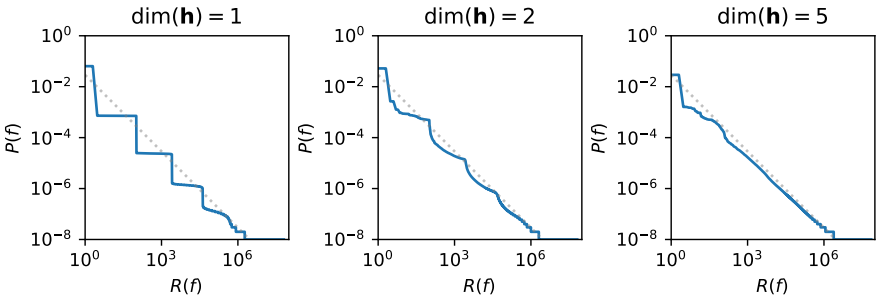


Figure F.10: More learnable eigenmodes lead to a smoother relation between $P(f)$ and $R(f)$ with $m = 50$. When $\dim(\mathbf{h}) = 1$, the relation between $P(f)$ and $R(f)$ is close to a staircase. By $\dim(\mathbf{h}) = 2$, the staircase is mostly washed out into a smooth curve, and by $\dim(\mathbf{h}) = 5$ the curve is almost entirely smooth aside from the usual artifacts at very low rank.

large eigenvalues of size 5). The eigenvectors are orthogonal. We then sample 10^8 times from a GP using this kernel.

F.8.3 Zipf's law for equicorrelated kernels

In the previous section, we showed that kernels have Zipfian priors when their eigenmodes can be separated into a small (relative to the number of samples) "learnable" set and a large "unlearnable" set. In this section, we provide some exact results for the equicorrelated kernel, which has one learnable eigenvector.

$$K_{ij} = \delta_{ij} + (1 - \delta_{ij})\rho, \tag{F.123}$$

where $\rho < 1$. Infinite-width neural network kernels are equicorrelated in the infinite-depth limit Appendix F.8.6, and may be rescaled to have variance 1 as above.

In the infinite-width limit, the network output preactivations y_i at initialization for m fixed inputs are sampled from $\text{NORMAL}(0, K)$. We can represent the outputs as

$$y_i = \sqrt{\rho}Y + \sqrt{1 - \rho}z_i, \tag{F.124}$$

where $Y, z_i \sim \text{NORMAL}(0, 1)$. Note that due to the permutation symmetry of this particular distribution over y_i's, the probability of a function f is determined only by the number of 0's, which we call t, and the number of 1's, equal to $m - t$ [263].

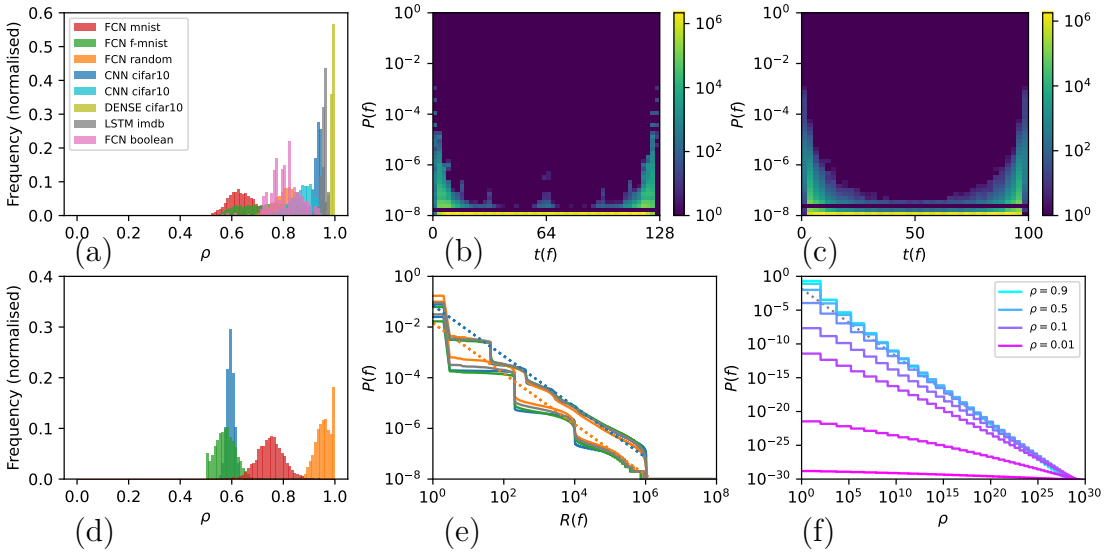


Figure F.11: Extending the equicorrelated kernel to actual kernels. (a) Histogram of $\rho_{ij} = \frac{K_{ij}}{\sqrt{K_{ii}K_{jj}}}$ for $i \neq j$ for NNGP kernels with $m = 100$. All architectures and datasets (from Fig. 5.1) have $\min(\rho_{ij}) > 0.5$. (b) FCN on $n = 7$ boolean data ($m = 2^7$) (c) FCN on Fashion-MNIST ($m = 100$), showing the density of functions for a given $t(f)$ v.s. $P(f)$. The most likely functions are generally the low-entropy functions which would be most likely for the equicorrelated kernel. For the boolean data, however, the bias is clearly not solely due to entropy bias. (d) shows the K_{ij} for $i \neq j$ for kernels with $m = 100$, generated by randomly sampling off-diagonal elements from distributions with fixed means $0.5 \leq \mu \leq 1$ and standard deviations $0.05 \leq \sigma \leq 0.25$, with any sample truncated at 0.5 and 1. Matrices that were not valid kernels (due to negative eigenvalues) were rejected. These kernels produced the rank-probability plot (e), with the colour schemes consistent across the two figures. The top group of lines shows $m = 20$, and the bottom group $m = 100$. (f) The entropy-bias only zipf's law curves with different ρ (using Eq. (F.127))

Since the classification of a given output is based only on its sign, we have,

$$P(f) = P\left(y_1 < 0, ..., y_t < 0, y_{t+1} > 0, ..., y_m > 0\right) \tag{F.125}$$

$$= \int_{-\infty}^{\infty} dY P(Y) \prod_{i=1}^{t} \left[P\left(z_i < -\sqrt{\frac{\rho}{1-\rho}} Y \mid Y \right) \right] \prod_{i=t+1}^{m} \left[P\left(z_i > -\sqrt{\frac{\rho}{1-\rho}} Y \mid Y \right) \right] \tag{F.126}$$

$$= \int_{-\infty}^{\infty} dY \frac{e^{-Y^2/2}}{\sqrt{2\pi}} \Phi\left(\sqrt{\frac{\rho}{1-\rho}} Y \right)^{m-t} \left(1 - \Phi\left(\sqrt{\frac{\rho}{1-\rho}} Y \right) \right)^{t}, \tag{F.127}$$

with the cumulative density function of the Gaussian distribution defined as:

$$\Phi(x) = \int_{-\infty}^{x} dx' \frac{e^{-x'^2/2}}{\sqrt{2\pi}} = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}} \right) \right) \tag{F.128}$$

Firstly, we note that this is a special case of the class of models considered

in [108], in particular a model of noninteracting Ising spins with a non-Gaussian distribution of external fields. Defining a "magnetic field" $h(Y)$ as

$$h(Y) \equiv \frac{1}{2} \log \left(\frac{P(y_i > 0|Y)}{P(y_i < 0|Y)} \right), \tag{F.129}$$

the model maps to

$$P(f) = \int \mathrm{d}h P(h) e^{-\frac{1}{2}m(h\alpha(f) - \log\cosh h}. \tag{F.130}$$

A similar mapping can be carried out if the final classification is made on the basis of a different nonlinearity, e.g. a random classification using a softmax.

The general arguments of [108] then establish that if $m \to \infty$ at fixed k (i.e., $m, r \to \infty$), Zipf's law will be observed, and that at finite rank r, as $m \to \infty$, Zipf's law will be observed if the distribution of h (equivalently, the distribution of k) is "sufficiently wide".

Condition on ρ for Zipf's law

How large must ρ be for Zipf's law to hold for all ranks? Firstly, note that this integral can be performed analytically for $\rho = 1/2$ using integration by parts to form a recurrence relation. The result is

$$P(f) = \frac{1}{m+1} \binom{m}{t}^{-1}. \tag{F.131}$$

By definition of $S(t)$ we have

$$E(t) = S(t) + o(m). \tag{F.132}$$

Thus, we expect to see Zipf's law for $\rho = 1/2$.

For general ρ, the calculation of $P(f)$ for large m can be done by the saddle-point method, as in [108]. We use the definition $t = km$, where $k \in [0, 1]$ denotes the fraction of 0's. We make the substitution $z = \sqrt{\rho/(1-\rho)}Y$ to get,

$$P(f) = \int_{-\infty}^{\infty} dz \sqrt{\frac{1-\rho}{2\pi\rho}} e^{-(1-\rho)z^2/2\rho} e^{m[(1-k)\ln\Phi(z) + k\ln(1-\Phi(z))]}, \tag{F.133}$$

$$= \int_{-\infty}^{\infty} dz Q(z) e^{m[(1-k)\ln\Phi(z) + k\ln(1-\Phi(z))]}. \tag{F.134}$$

We will call z the latent variable, and we define the prior distribution over z, $Q(z) = \sqrt{\frac{1-\rho}{2\pi\rho}} e^{-(1-\rho)z^2/2\rho}$.

Since the exponent in equation Eq. (F.134) is extensive in m, the leading-order behaviour in m is given Laplace's method (the saddle-point approximation), by noting that the main contribution to the integral comes from the maximum at $z = z^*(k)$, where

$$\frac{\partial}{\partial z}\left[(1-k)\ln\Phi(z) + k\ln\left(1-\Phi(z)\right)\right]\Big|_{z=z^*(k)} = 0, \tag{F.135}$$

$$\Rightarrow z^*(k) = \Phi^{-1}(1-k). \tag{F.136}$$

The full saddle-point approximation for the integral in Eq. (F.134) is

$$P(f) \approx \frac{Q(z^*)}{\sqrt{2\pi\left|L''(z^*)\right|}} e^{mL(z^*)}, \tag{F.137}$$

where $L(z) = (1-k)\ln\Phi(z) + k\ln(1-\Phi(z))$.

Firstly, if we take $m \to \infty$ at fixed k, the non-exponential factors can be neglected. We have

$$\ln P(f) = m((1-k)\ln(1-k) + k\ln k) + O\left(\ln m\right), \tag{F.138}$$

$$= -ms(f) + O\left(\ln m\right), \tag{F.139}$$

where $s(f) \equiv (1-k)\ln(1-k) + k\ln k$ is the entropy rate of f when it is treated as a binary string. Crucially, the multiplicity of functions with k is indeed $e^{S(k)} = e^{ms(k)}$. We thus again have $E = S + o(m)$.

On the other hand, recall that the full saddle-point approximation for the integral in Eq. (F.134) is

$$P(f) \approx \frac{Q(z^*)}{\sqrt{2\pi\left|L''(z^*)\right|}} e^{mL(z^*)}, \tag{F.140}$$

where $L(z) = (1-k)\ln\Phi(z) + k\ln(1-\Phi(z))$. To obtain Zipf's law, we saw above that we need to be able to neglect the factors outside the exponential. Thus, deviations from Zipf's law will be observed for ranks where this prefactor varies substantially.

Empirically, we observe that deviations caused by $Q(z^*)$ are larger than the deviations caused by $|L''(z^*)|$. Thus, we consider $Q(z^*)$. Since the argument of [108]

establishes that Zipf's law holds exactly as $m, r \to \infty$, we expect to see the strongest deviations from Zipf's law at low ranks. Neglecting the two most likely functions (for which Zipf's law never holds exactly), we consider the next m most likely functions, which have $k = 1/m$, whose probability we denote p. If Zipf's law holds exactly, then from the normalization (Appendix F.1.2) we expect that $p \propto 1/m$.

Letting $\tilde{Q}$ denote $Q(z^*)/Q(0)$ for these m functions, we have

$$\tilde{Q} = e^{-\frac{1-\rho}{2\rho} z^{*2}}. \tag{F.141}$$

.

For small k, $\Phi^{-1}(1-k) \sim \sqrt{-2\log k}$. Thus,

$$\tilde{Q} \sim e^{-\frac{1-\rho}{\rho}\log m} \sim m^{-\frac{1-\rho}{\rho}}. \tag{F.142}$$

.

For $\rho < 1/2$, this decays faster than p must decay under Zipf's law. Thus, for $\rho < 1/2$ we expect to see deviations from Zipf's law, at least at low ranks.

F.8.4 Experimental results

Fig. 5.2 studies the effect of moving 10-layer tanh-activated FCNs into the chaotic regime, by increasing σ_w. It shows histograms of the off-diagonal kernel elements $K_{ij}/\sqrt{K_{ii}K_{jj}}$ and the corresponding plots $P(f)$ v.s. $R(f)$. $\sigma_w = 0.75, 1.0$ has a prior close to Zipf, and both generalise equally well. For $2 \leq \sigma_w \leq 5$, we move away from Zipf with α significantly lower than 1, and generalisation is worse.

Fig. F.11 shows related experiments. (a) shows histograms for the kernels of architectures found in Figs. 5.1 and F.4, all of which have $\rho_{ij} > 0.5$. Note that Fig. F.7 demonstrates that it is possible for Zipf's law to be seen even when some ρ_{ij} are below 0.5. Panels (b) and (c) show the probability $P(f)$ of functions in two different neural networks as a function of $t(f)$ which controls the probability of a function for the equicorrelated kernel. We see that most of the most likely functions are precisely those which would be most likely for the equicorrelated kernel: "low-entropy" functions where most labels agree. There are, however,

some deviations—for example, for the 2 hidden layer FCN on the boolean data, some maximum entropy functions have very high $P(f)$. Panel (d) shows a series of artificially generated kernels with $\rho_{ij} > 0.5$, and (e) their priors, which are Zipfian, but significantly blockier than the DNN-generated priors. Finally, (f) shows the theoretical curves for the equicorrelated kernels, for different values of the off-diagonal element ρ.

F.8.5 How close are NNGP kernels to equicorrelated?

For l-layered ReLU networks with biases set to 0, the value of $K^l_{ij}/\sqrt{K^l_{ii}K^l_{jj}} \equiv a_l$ is given by the recurrence relation,

$$a_l = \frac{1}{\pi}\left(\sqrt{1-a_{l-1}^2} + (\pi - \arccos a_{l-1})\, a_{l-1}\right). \tag{F.143}$$

Note that $a_l > a_{l-1}$ for all $a_{l-1} < 1$ ($a_l = 1$ is an unstable fixed point), so that the lower bound for a_l is given by the sequence generated with $a_0 = -1$, which gives,

$$\left\{\min\left(\frac{K^l_{ij}}{\sqrt{K^l_{ii}K^l_{jj}}}\right)\right\}_{l\in\mathbb{R}^\kappa} = \{-1, 0, 0.318, 0.494, 0.605, ...\}\,. \tag{F.144}$$

Eq. (F.144) shows that values of ρ larger than ~ 0.5 are achieved already after the third layer for any type of input data (other than an orthogonal set), so FCNs with ReLU activations will lead to Zipf's law, provided experiments Fig. F.11(a) are general.

The value of ρ when approximating the distribution as equicorrelated can be chosen as the mean of the off-diagonal normalised kernel matrix elements, $\rho \sim \left\langle K_{ij}/\sqrt{K_{ii}K_{jj}}\right\rangle$ at each layer. Fig. F.11(a) shows that the off-diagonal elements of NNGP kernels for standard architectures (CNNs, FCNs, LSTMs) on standard datasets (Fashion MINST, CIFAR10, IMDb) are all greater than 0.5, but the variance can be quite large.

However, this instance of Zipf's law arises due to an entropy bias and is not sufficient to explain Zipf's law qualitatively for practical neural networks. This is because there can be a large range in $P(f)$ for functions with the same entropy – for example, in the boolean system structured high entropy functions like 010101 . . .

are more probable than many low entropy functions. This structure is also the most crucial for generalisation, as often we want to distinguish between functions with the same (or similar) entropy. However, this approximation does seem to work well for chaotic DNN kernels, see Fig. 5.2.

F.8.6 Infinite depth chaotic tanh-activated DNNs have a uniform prior

The chaotic regime of deep neural networks was studied, for example, in [79, 264]. The most relevant result for our purposes concerns the correlations between activations at l layers, for neural networks at initialisation, taking the mean field approximation (where preactivations z_i^l are replaced by a Gaussian whose first two moments match those of z_i^l). This mean field approximation is exact for infinitely wide neural networks.

Specifically, it is shown in [79] that for neural networks with tanh activations and combination of σ_w and σ_b in the "chaotic regime" (for example, $\sigma_w = 2.5$, $\sigma_b = 0.0005$),

$$\mathbb{E}[z_i^l(x_a)z_j^l(x_b)] \xrightarrow{l\to\infty} q^*\delta_{ij}(\delta_{ab} + c_{ij}(\sigma_w, \sigma_b)) \tag{F.145}$$

where x_a and x_b are inputs to the network, and z_i^l is the i'th preactivation in the l'th layer. $c_{ij}(\sigma_w, \sigma_b)$ is a correlation term between inputs i and j, and it is further shown that $(\sigma_w, \sigma_b) \xrightarrow{\sigma_w\to\infty} 0$ for any fixed σ_b. By contrast, neural networks in the ordered regime (characterised by a small σ_w and large σ_b) have an attractive fixed point at 1 – all inputs become perfectly correlated.

For a neural network with $L \to \infty$ infinitely wide hidden layers and a single output neuron, Eq. (F.145) can be used with $i = j = 1$, and $f(x_a)$ the final layer's output for input x_a to give

$$K(x_a, x_b) = \mathbb{E}[f(x_a)f(x_b)] \xrightarrow{l\to\infty} q^*\delta_{ab},$$

which defines the kernel used in Neural Network Gaussian Processes (NNGPs). The prior probability of an output y, $P(y)$ is given by

$$P(y) \propto \exp\Big(-y^T K^{-1} y\Big),$$

where K is the kernel matrix for m inputs, and y the output vector (pre-thresholded) for the m outputs. As K^{-1} is a diagonal matrix, all predictions are uncorrelated and have an equal chance of being 0 or 1, so the probability of any thresholded function f must be uniform:

$$P(f) = 2^{-m}.$$

A similar result was proved in [249] but for NTK – NTK in the limit of infinite depth converges to an equicorrelated matrix with correlation $c = c(\sigma_w, \sigma_b)$, with the same limiting behaviour as the NNGPs.

F.9 Structured priors can produce superzipfian power laws

The modern machine-learning models shown in the main text always seem to produce zipfian or subzipfian initialisation priors, although they may sometimes produce superzipfian effective priors.

One straightforward way to produce a superzipfian prior is to consider a *structured* prior, as in, for example, Bayesian Model Selection. We consider a sequence of model classes, $F_1, F_2, \ldots$, of increasing complexity. For example, we may characterize complexity by the VC dimension d, and have $d_1 < d_2 < \ldots$. We then assign a finite weight $P(F_i)$ to each class.

It is well-known [213, 265] that such a prior produces a bias against complexity, through effective "Occam factors" which penalize more complex models (since the weight $P(F_i)$ is spread over exponentially more functions when d_i is larger).

In particular, we may make a rough estimate by assuming that the number of functions f on a particular dataset of size m saturates the bound induced by the VC dimension, i.e.,

$$N(F_i) = \sum_{k=1}^{d_i} \binom{m}{k} \sim 2^{mH(d_i/m)}. \tag{F.146}$$

In this case, a flat prior on the class i gives us

$$\log P(f) \approx \log P(F_i) - \log N(F_i) \tag{F.147}$$

$$E = S + o(m), \tag{F.148}$$

which gives Zipf's law, as discussed in Appendix F.8.

It is clear from inspecting the above equation and considering the arguments discussed in Appendix F.8 that we can get a superzipfian prior by choosing $\log P(F_i)$ to decay with d_i in an m-independent way (or equivalently, to decay with m at fixed ratio d_i/m). For example, we can achieve this by a prior

$$P(F_i) \propto e^{-ad_i}. \tag{F.149}$$

To test this idea, we compute numerically the prior of a simple classification model using trigonometric polynomials. The model is

$$f_i = \text{sign} \sum_{k=0}^{K-1} A_k \cos(2\pi k x + \varphi_k) \tag{F.150}$$

We draw x uniformly and independently on the interval $[0, 1]$.

At a given K, this function class has VC dimension $2(K-1)+1$, because the sign of a trigonometric polynomial of order n may change at most $2n$ times.

Within a value of K, we put uniform priors on φ_k , and draw A_k from a standard normal distribution. We then put the prior $P(K) \propto e^{-aK}$ on K.

As expected, this model produces a supzerpfian prior, as shown in Fig. F.12. The relationship between the exponent and a is difficult to predict analytically, however, because this function class turns out to not saturate the bound on the number of observed functions for a given VC dimension.

F.10 Lemma on sums of kernel eigenvalues

We want to establish conditions under which, for a bounded kernel with true kernel eigenvalues $\Lambda_\mu \sim 1/\mu^\ell$ and the gram matrix with eigenvalues λ_μ constructed from m samples:

$$\sum_{\mu=\mu_1}^{\mu_2} \lambda_\mu \approx m \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu, \tag{F.151}$$

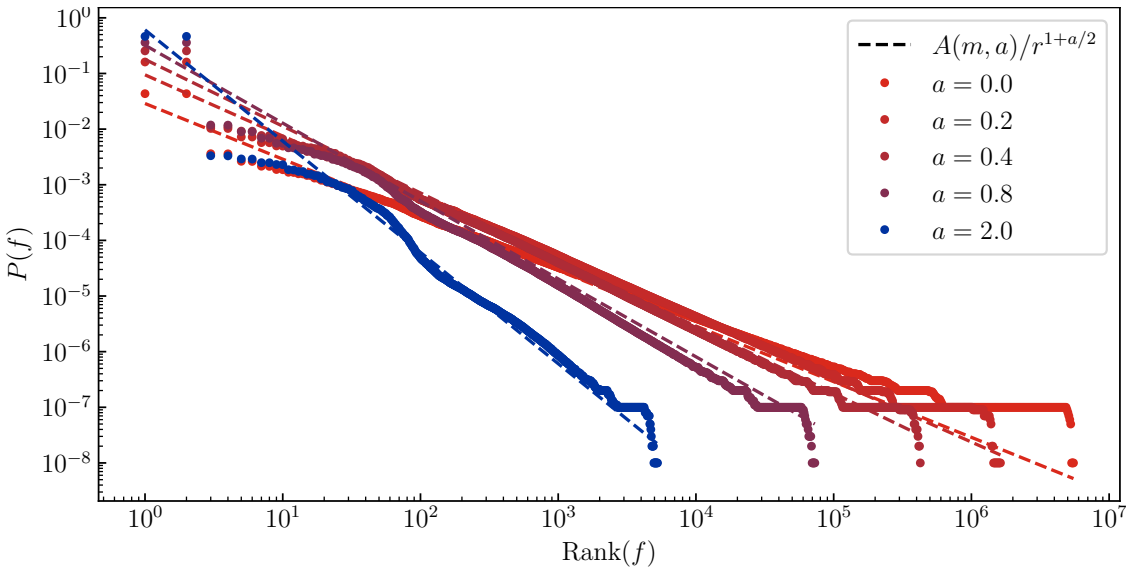


Figure F.12: Bayesian model selection, with the probability assigned to classes decreasing exponentially with their VC dimension, can produce priors ranging from superzipfian to Zipfian. Details as described in the text.

or more precisely

$$\frac{\left|\sum_{\mu=\mu_1}^{\mu_2} \lambda_\mu - m \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu\right|}{m \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu} \to 0. \tag{F.152}$$

For individual eigenvalues of a bounded kernel, for any $1 \leq r \leq m$ [262],

$$|\lambda_\mu - m\Lambda_\mu| = O\left(r^2 \Lambda_\mu \Lambda_r^{-1/2} + m \sum_{\nu>r} \Lambda_\nu + \sqrt{m}\sqrt{\sum_{\nu>r} \Lambda_\nu}\right). \tag{F.153}$$

The total error on the sum is then

$$\frac{\left|\sum_{\mu=\mu_1}^{\mu_2} \lambda_\mu - m \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu\right|}{m \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu} = O\left(\frac{r^2}{m\sqrt{\Lambda_r}} + \frac{\sum_{\nu>r} \Lambda_\nu}{\sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu} + \frac{\sqrt{\sum_{\nu>r} \Lambda_\nu}}{\sqrt{m} \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu}\right) \tag{F.154}$$

Take $\mu_1 = m^a$, $\mu_2 = m^b$, and $r = m^c$. This is then

$$\frac{\left|\sum_{\mu=\mu_1}^{\mu_2} \lambda_\mu - m \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu\right|}{m \sum_{\mu=\mu_1}^{\mu_2} \Lambda_\mu} = O\left(m^{(2+\frac{\ell}{2})c-1} + m^{(\ell-1)(a-c)} + m^{(\ell-1)(a-\frac{c}{2})-\frac{1}{2}}\right). \tag{F.155}$$

The first two terms go to zero as $m \to \infty$ as long as $a < c < \frac{1}{2+\ell/2}$. For $c < 1/(2+\ell/2)$, the second term goes to zero as long as $2a < \frac{1}{2+\ell/2} + \frac{1}{\ell-1}$.

F.11 Predictive information and Zipf's law

As mentioned in the main text, here we show that the predictive information of a sequence *sampled* from a power law prior only diverges as $m \to \infty$ for a Zipfian

($\alpha = 1$). This offers a hint that non-Zipfian priors must be bad for learning—if they were correct priors, it would imply that only a finite amount of information about the true function can ever be learned, even with infinite data!

F.11.1 The predictive information

Consider a sequence of observations $\{z_i\} \equiv \{x_i, y_i\}_{i=1}^{\infty}$ drawn from some generating distribution $P(\{z_i\})$. Let $\mathbf{x}, \mathbf{y}, \mathbf{z}$ refer to a length-m subset of this sequence,$\mathbf{z} = \{z_i\}_{i=1}^{m}$. The entropy of $\mathbf{z}$ is extensive; as $m \to \infty$, $S(\mathbf{z}) \to sm$. The amount of information that can be learned about the future from a finite number of observations is quantified by the *predictive information*, which one can show is given by the subextensive part of the entropy [114]:

$$I_{\text{pred}}(m) \equiv I(\mathbf{z}, \{z_i\}_{i=m+1}^{\infty}) = S(\mathbf{z}) - sm. \tag{F.156}$$

The predictive information is necessarily subextensive, but when there is something meaningful to learn from the data about the future, it still diverges as $m \to \infty$. For simple generative models with k learnable parameters, $I_{\text{pred}}(m) \sim \frac{k}{2} \log N$, while for models with an infinite number of parameters it grows like a power law [114].

F.11.2 Predictive information for power-law distributions in function space

Assuming that the inputs x_i are drawn i.i.d. and the labels are then generated from them via some function, $y_i = f(x_i)$, we have

$$S(\mathbf{z}) = S(\mathbf{x}) + S(\mathbf{y}|\mathbf{x}) = ms_x + S(\mathbf{y}|\mathbf{x}). \tag{F.157}$$

Thus, the distribution of $\mathbf{x}$ makes no contribution to the predictive information, which is determined by the subextensive part of the conditional entropy $S(\mathbf{y}|\mathbf{x})$.

Now, let us assume that for all typical sequences $\mathbf{x}$, $P(\mathbf{y}|\mathbf{x})$ has a power-law form, $P(\mathbf{y}|\mathbf{x}) = P_R \equiv c(m)/\operatorname{rank} \mathbf{y}|\mathbf{x}^{\alpha}$. We then have

$$S(\mathbf{y}|\mathbf{x}) = -\left\langle \sum_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \log p(\mathbf{y}|\mathbf{x}) \right\rangle_{\mathbf{x}}, \tag{F.158}$$

and thus

$$S(\mathbf{z}) = \text{extensive stuff} - \sum_{R=1}^{m} P_R \log P_R. \tag{F.159}$$

We approximate this sum by an integral,

$$-\sum_{R=1}^{m} P_R \log P_R \approx -\int \frac{r^{-\alpha}}{\int_1^m (r')^{-\alpha}\,\mathrm{d}r'} \log\left(\frac{r^{-\alpha}}{\int_1^m (r')^{-\alpha}\,\mathrm{d}r'}\right) \mathrm{d}r \tag{F.160}$$

A straightforward computation then gives:

$$I_{\text{pred}}(\alpha, m) \sim \begin{cases} \log m, & \alpha = 1 \\ O(1), & \alpha \neq 1. \end{cases} \tag{F.161}$$

F.12 Fat tails: basics of Zipf's law rank plots

Here, we recapitulate some basic properties of rank-ordered frequency plots with power-law forms that have a maximum of N_f distinct outcomes in them. In Eq. (F.2) we use an integral approximation to determine the constant A in $P(f) = AR(f)^{-\alpha}$ with $\alpha = 1$. We use the same approach for $\alpha \neq 1$, with $N_f = C^m$, where C is the number of classes.

$$1 \approx \int_1^{N_f} \frac{A}{r^\alpha} dr = \frac{A}{1-\alpha}\left[1 - N_f^{(1-\alpha)}\right]. \tag{F.162}$$

In the rest of this section, we write $\alpha = 1 + \delta$, so $A \approx \delta/(1 - N_f^{-\delta}) = \delta/(1 - C^{-m\delta})$. The cumulative probability where a fraction $0 < x \leq 1$ of the total probability weight is reached by all functions of rank $\leq r_x$ is given by $r_x = \left(1 - x + xC^{-m\delta}\right)^{-1/\delta}$. For our analysis, it is interesting to note how differently these quantities behave if $\delta = 0$, $\delta > 0$ or if $\delta < 0$. Below we assume large N_f, which roughly means that $\ln N_f \gg 1/\delta$. Then,

- For $\delta = 0$:

$$A \approx 1/\ln N_f = 1/(m \ln C), \qquad r_x = N_f^x.$$

  The scale A drops logarithmically in N_f. The rank for cumulative probability x grows as a power of N_f. For example, the rank for 50% of the total cumulative probability is $r_{50\%} = \sqrt{N_f} = C^{m/2}$.

- If $\delta > 0$, then:

$$A \approx \delta, \qquad r_x \approx 1/(1-x)^{1/\delta}.$$

  To leading order in N_f, A and r_x are independent of N_f.

- If $\delta < 0$, then:

$$A \approx |\delta|/N_f^{|\delta|}, \qquad r_x \approx x^{1/|\delta|} N_f.$$

  To leading order in N_f, A drops exponentially with increasing δ, and r_x is directly proportional to N_f.

To get a sense of how much a small change in δ matters, consider the $n = 7$ Boolean dataset where, $N_f = 2^{128}$ for all $m = 128$ inputs. Then

- If $\delta = 0$, then we have $A \approx 0.011$, and $r_{50\%} \approx 1.8 \times 10^{19}$, at which point $P(f) \approx 1.2 \times 10^{-21}$.
- If $\delta = 0.1$, then $A \approx 0.1$, and $r_{50\%} \approx 1024$, at which point $P(f) \approx 10^{-5}$.
- If $\delta = -0.1$, then $A \approx 1.4 \times 10^{-5}$, and $r_{50\%} \approx (1/1024) \times 2^{128} = 2^{118}$ at which point $P(f) \approx 10^{-41}$.

In other words, under the ansatz of a power law all the way down, tiny differences in the exponent can lead to enormous differences in some key properties of the distribution of $P(f)$.

We can also analyse what happens if it is perfectly Zipf up to 10^6 (that we roughly observe), and then crosses over to a different power for unobserved ranks. One continuous implementation of this is:

$$P(f) = \begin{cases} \frac{C(m,\alpha)}{r}, & r < r_0 \\ \frac{C(m,\alpha) r_0^{\alpha-1}}{r^\alpha}, & r_0 \leq r \leq 2^m. \end{cases} \tag{F.163}$$

As shown in Fig. F.13, the normalization of this ansatz, with $r_0 = 10^6$, $m = 100$, is less strongly affected by α than the pure power ansatz, but is still fairly sensitive to α.

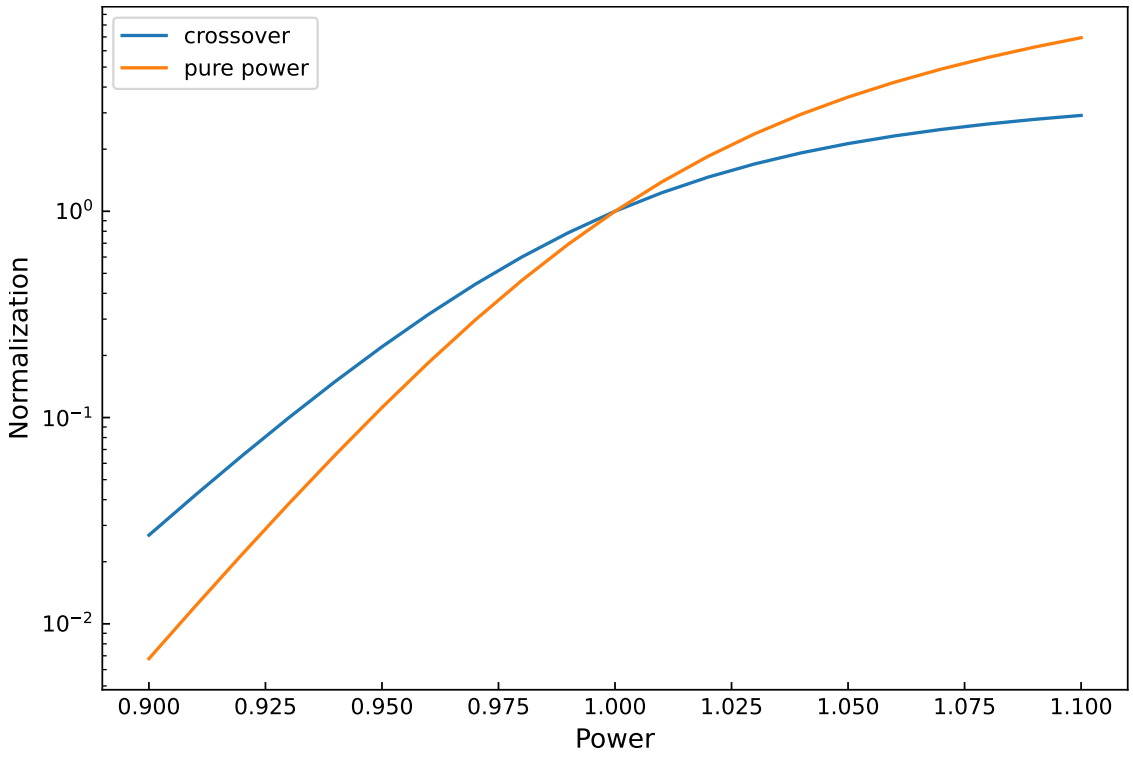


Figure F.13: Comparison of normalization for pure power $\alpha \neq 1$ vs. a crossover from Zipf to $\alpha \neq 1$.

G

Appendices for Chapter 6

G.1 Approximating the eigenfunctions and eigenvalues in practice

To calculate the eigenfunctions and eigenvalues using Eq. (6.3) for a given parameter-function map Φ, we also need the true input distribution q, which is in principle inaccessible. However, we assume that for a large enough training set $S_{tr} \in \mathcal{X}$ we can use its n datapoints to obtain n feature vectors of size p, and that these provide a good enough estimate of q that we can use the discrete Nyström method [266, 121] to extract eigenvalues and eigenfunctions from the integral equation Eq. (6.3) with sufficient accuracy. We perform singular value decomposition (SVD) on the feature matrix $\Phi(S_{tr}) \in \mathbb{R}^{p \times n}$ to obtain singular values $s \in \mathbb{R}^p$ and p left column eigenvectors $u_k \in \mathbb{R}^p$. We can approximate empirical eigenvalues of T by squaring s and using the left column eigenvectors $u_k \in \mathbb{R}^p$, s_k, and feature map $\Phi : \mathcal{X} \to \mathbb{R}^p$ to obtain k^{th} eigenfunction $e_k(x) = u_k^T \Phi(x)/s_k$. Note that the eigenfunctions can be used on any data point beyond the training set. Also, we have implicitly assumed $n > p$, but when $n < p$, we can only approximate up to n eigenfunctions and eigenvalues. The algorithm is summarized in Algorithm 12.

To calculate the inner product in Eq. (6.7) involving f^* or $\hat{f}$ in Eq. (6.10) we cannot use the training set, because if one trains to zero training error, both functions

Algorithm 12 Empirical eigenfunctions and eigenvalues

Φ, S_{tr}, x
$\varphi \leftarrow \Phi(S_{tr})$ (forward transform the training set into the feature space)
$u, s, v \leftarrow SVD(\varphi)$ (find the eigenvalues and eigenvectors via SVD)
$[u_1, u_2, \cdots, u_p] \leftarrow u$ (find the column vectors of u)
for k in $1, 2, \ldots, p$ **do**
    $\rho_k \leftarrow s_k^2$ (approximate eigenvalues)
    $e_k(x) \leftarrow u_k^T \Phi(x)/s_k$ (approximate eigenfunctions)
end for

will be equivalent on S_{tr}. So instead, given the eigenfunctions, we use the test set S_{te} to approximate the relevant inner products as follows:

$$\frac{\langle e_k|f_j\rangle^2}{\|e_k\|_2^2\|f_j\|_2^2} = \frac{(\int_\chi e_k(x)f_j(x)q(x)dx)^2}{\int_\chi e_k(x)^2q(x)dx\int_\chi f_j(x)^2q(x)dx} \approx \frac{\left(\sum_{x^{(i)}\in S_{te}} e_k(x^{(i)})f_j(x^{(i)})\right)^2}{\sum_{x^{(i)}\in S_{te}} e_k(x^{(i)})^2\sum_{x^{(i)}\in S_{te}} f_j(x^{(i)})^2}. \tag{G.1}$$

In Appendix G.5, we discuss the validity and limits of our approximation methods in more detail. Briefly, to check how well our approximation to q works we used significantly smaller subsets of the training set and found that for scenarios we typically care about, where only a subset of all the eigenfunctions are relevant, this makes no meaningful difference to our measures. For large training and test sets (e.g. $n_{train} = 5 \times 10^4$ and $n_{test} = 10^4$ for CIFAR10) the approximation of the true distribution q with a finite sample as done in Algorithm 12 and Eq. (G.1) should be fairly accurate. The Nyström method becomes less accurate for eigenfunctions with relatively small eigenvalues ρ_k (typically for large k). Furthermore, cumulative projection measures such as $\Pi^*(k)$ suffer from finite size errors when $k \approx n_{test}$ because we use S_{te} in Eq. (G.1). Neither of these sources of error is typically that important for two reasons. Firstly, we normally work in the limit $p \ll n_{test}$. Secondly, we will see that typically a relatively small number $k \ll p$ eigenfunctions dominate our measures. Nevertheless, due to these finite size effects, we exercise caution and present plots only up to $k \sim \mathcal{O}(10^3) \ll n_{test}$.

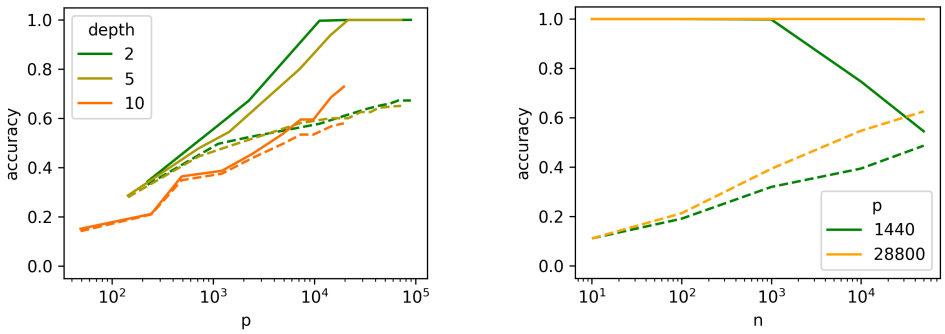
G.2 DNN performance for a fixed feature map as a function of width

In this section, we expand on the experiment in Fig. 6.3(b) where we explore the width p needed for a DNN with a frozen feature map (random features) to reach (near) zero training error. We compare a CNN, ResNet18 and VGG16 on both CIFAR10 and CIFAR100. In Fig. G.1 we observe that to first order, a width on the order of $p \approx n$ is needed to train a CNN to zero training error on CIFAR10. To second order, this width depends on depth as well, with larger widths needed for deeper CNNs. We explore a much smaller range of widths for ResNet18 and VGG16 architectures for CIFAR10 in Fig. G.2, and for CIFAR100 in Fig. G.3. We observe that if the DNN is trained with SGD, both these architectures can reach zero training error for very narrow final layer widths p, but much larger widths are needed for DNNs with a frozen feature map.

In the main text, we did not study the effects of the widths of the intermediate layers. One could imagine the extreme case of a bottleneck layer inserted just before the penultimate layer, that would greatly reduce expressivity. Here we keep the overall width constant, but it would be interesting to vary the width of intermediate layers.

In more detail: Fig. G.1 shows a CNN with kernels of size 3×3, a max-pooling layer with kernel size 2×2, and a final linear layer on the flattened output. The output dimension is controlled by the size of this flattened output, meaning that the number of filters per layer determines p. When the last layer has $p = 90000$ for the depth 2 CNN, we have 500 filters per layer (increasing depth decreases p slightly for a fixed number of filters as the image is shrunk by 2 pixels on each dimension with each convolutional layer). Depth counts the number of matrices in the CNN (so a single convolutional layer plus the final layer has depth 2). Fig. G.1(a) shows that $p \sim 10^4$ is enough to fit the training data for the depth 2 and 5 models, but for depth 10 and for the maximum width, the model cannot do better than 70% training accuracy. Note also that even when 100% training accuracy can be

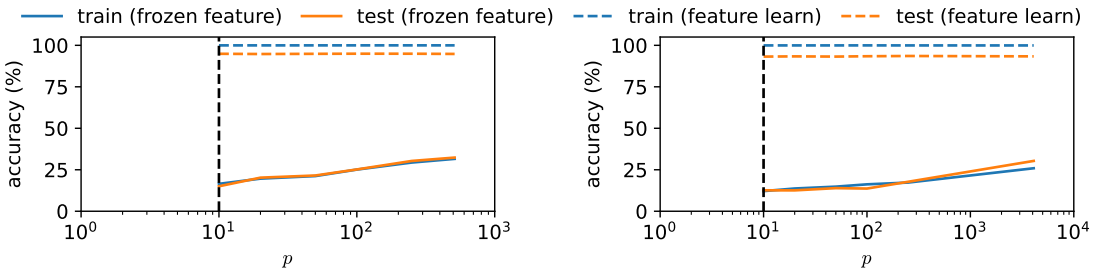
achieved, generalisation continues to improve with increasing width for the depth 2 and 5 models. Fig. G.1(b) explores the generalisation error with training set size for the depth 5 model at two different widths. It illustrates how narrower frozen feature models can fit less training data to 100% accuracy, as expected.



(a) CNN on CIFAR10 as a function of width

(b) CNN on CIFAR10 as a function of training set size

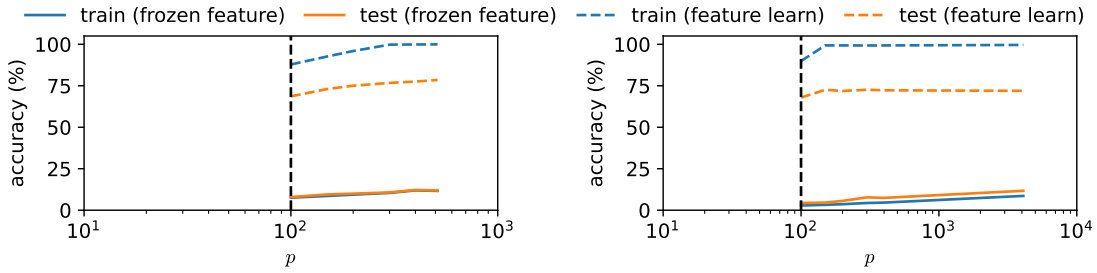
Figure G.1: CNN performance for frozen layers (a) The training error (solid) and test error (dashed) for CNNs with a frozen feature map and different depths, trained on CIFAR10, as a function of the intermediate/last layer widths p. The training set size is fixed at the standard $n = 50000$. (b) Training error (solid) and test error (dashed) for a 5-layer CNN with widths $p = 1440$ and $p = 28800$, trained on different training set sizes n. Note that to first order, the training error reaches zero for widths on the order of the training set size for this data set.



(a) ResNet18

(b) VGG16

Figure G.2: Performance by the width of the input for the last layer for CIFAR10 Two widely used models (a) ResNet18 and (b) VGG16 are trained on CIFAR10 with two different methods. The models were trained with SGD, allowing full feature learning (solid), or trained with all layers except the last layer fixed. Performance is plotted across various widths of the last layer, sampled between its original width and $C = 10$. First, we observe that feature learning models outperform last layer-only trained models by a significant margin. Furthermore, the performance of feature learning models remains independent of the width of the last layer, in contrast to the correlation observed in last layer-only trained DNNs.



(a) ResNet18 (b) VGG16

Figure G.3: Performance by the width of the input for the last layer for ResNet18 and VGG16 on CIFAR100 (a) ResNet18 and (b) VGG16 on CIFAR100. Similar results are obtained in Fig. G.2, but the performance of feature learning models now changes more with the width than it does for CIFAR10. This difference can be attributed to the difficulty that intermediate layers face in learning features for a more challenging task.

G.3 On the definition of features

In the main text, we defined the eigenfunctions, $e_k(x)$, and eigenvalues, ρ_k, of the operator $T[f](x')$. How do these eigenfunctions relate to the basis functions $\varphi_k(x)$? One way one can rewrite $f(x) = \sum_{k=1}^{p} \theta_k \Phi_k(x)$ in $|e_k\rangle$ is as:

$$f(x) = \sum_{j=1}^{j=p} \theta_j \left[\frac{1}{\sqrt{\rho_k}} \int dx' q(x') \Phi_j(x') e_k(x') \right]_{jk} \sqrt{\rho_k} \, |e_k\rangle \,, \tag{G.2}$$

Where U_{jk} will denote the matrix in the square brackets. We note that the rows of U can be written as $\sqrt{\rho_i} e_i(x) = (U_i)_j \Phi_j(x) = U_i^T \Phi(x)$, meaning

$$\delta_{ij} = \int e_i(x) e_j(x) q(x) dx = (U_i)^T \sqrt{\frac{\rho_j}{\rho_i}} \left[\frac{1}{\sqrt{\rho_j}} \int \Phi(x) e_j(x) q(x) dx \right] = \sqrt{\frac{\rho_j}{\rho_i}} (U_i)^T (U_j), \tag{G.3}$$

demonstrating that $U^T U = U U^T = I$, and thus U is orthogonal. This argument shows that

$$f(x) = \sum_{k=1}^{p} \left[U_{kj} \theta_j \right] \left[\sqrt{\rho_k} e_k(x) \right]$$

where U preserves the norm of θ. This would mean the norm of a perturbation $||\Delta\theta||_2 = ||\Delta\theta'||_2$ where $\theta' = U\theta$. This fact means gradient descent on θ' would find the same solutions as on θ. For more information on the importance of model parameterisation, see e.g. [267]. Given this property, it would arguably

be desirable to call $\sqrt{\rho_k}e_k(x)$ the features, as we could write $f(x) = \sum_{k=1}^{p} \theta'_k z_k$ where the features $z_k = \sqrt{\rho_k}e_k(x)$.

Furthermore, when the model is overparameterised, the solution found is strongly influenced by ρ_k – so incorporating ρ_k into the definition of features would mean two models with the same features would always find the same solutions. It would be entirely consistent to call features $\sqrt{\rho_k}\,|e_k\rangle$, and then study its two components separately. However, one could also argue that as the eigenfunctions $e_k(x)$ are correctly normalized, it would not make sense to call features rescaled eigenfunctions – two models with the same $e_k(x)$ but different eigenvalues ρ_k, ρ'_k would disagree on how much the k'th feature is present in the input. This also seems undesirable.

As a result of these issues, and given that our measures use the correctly normalized $e_k(x)$, or ρ_k separately, we find it easier to call $|e_k\rangle$ the features and ρ_k the associated eigenvalue.

It is also worth briefly discussing exactly how the eigenfunctions $e_k(x)$ relate to more intuitive pictures of features – as abstract properties of the input (for example, the edge of an object, or the object itself, in image data). It has been shown in e.g. [268] that early layers of CNNs do indeed pick up on edges, and later layers pick up on more abstract features (like chain links) [269]. It is at present less clear how to make similar interpretations of our final layer features.

G.4 Two lemmas

In this section, we provide the proofs of two lemmas that are used in the main text. Throughout, we assume an infinite number of data and the following MSE loss.

$$\mathbf{L} = \int_{\mathcal{X}} \left(f(x) - f^*(x)\right)^2 q(x) dx \tag{G.4}$$

Lemma 17. *The Gradient Flow (GF) dynamics of a linear model of the following form $f(x) = \sum_{i=1}^{p} \theta_i \Phi_i(x)$ is diagonalized into the following p independent equations:*

$$\langle f|e_k\rangle\,(t) = \langle f^*|e_k\rangle\,(1 - e^{-\eta\rho_k t}), \tag{G.5}$$

where ρ_k and e_k are the eigenvalues and eigenfunctions of the intergral operator T (Eq. (6.3)), and η is the learning rate.

Proof. By using the gradient descent equation ($\frac{d\theta_i}{dt} = -\eta\frac{d\mathbf{L}}{d\theta_i}$), we can write the change of f as

$$\frac{df}{dt}(x') = \sum_i^p \frac{d\theta_i}{dt}\Phi_i(x') = \sum_i^p -\eta\frac{d\mathbf{L}}{d\theta_i}\Phi_i(x') \tag{G.6}$$

$$= -2\eta\int_\chi (f(x) - f^*(x))\sum_i^p \frac{df}{d\theta_i}(x)q(x)dx\Phi_i(x') \tag{G.7}$$

$$= -2\eta\int_\chi (f(x) - f^*(x))\sum_i^p \Phi_i(x)\Phi_i(x')q(x)dx \tag{G.8}$$

$$= -2\eta T\left[f - f^*\right](x') \tag{G.9}$$

The equation mimics the form of a matrix differential equation. If we take the inner product of both sides with e_k,

$$\frac{d\left\langle e_k|f\right\rangle}{dt} = -2\eta\left\langle e_k|T\left[f - f^*\right]\right\rangle \tag{G.10}$$

$$= -2\eta\rho_k\left(\left\langle e_k|f\right\rangle - \left\langle e_k|f^*\right\rangle\right), \tag{G.11}$$

where we used that fact that T is symmetric operator and $T[e_k] = \rho_k e_k$. Solving the differential equation for $\langle e_k|f\rangle$ with initial condition of $\langle e_k|f\rangle(0) = 0$ leads to

$$\left\langle f|e_k\right\rangle(t) = \left\langle f^*|e_k\right\rangle(1 - e^{-\eta\rho_k t}). \tag{G.12}$$

□

Lemma 18. *When $T = T_{MP}$ for a DNN, there is one-to-one mapping between $\Phi(x)$ and $\hat{f}(x)$ on the support of q.*

Proof. The MP operator, by definition, maps from the expressed function space $\hat{\mathcal{H}}$ to $\hat{\mathcal{H}}$. The entries of the feature map $[\Phi_1, \cdots, \Phi_p]$ span $\hat{\mathcal{H}}$, and entries of the expressed function $([\hat{f}_1, \cdots, \hat{f}_C])$, by definition, also span $\hat{\mathcal{H}}$. Since $\Phi(x)$ and $\hat{f}(x)$ are related by a linear transform – ignoring the bias term as constant function is in $\hat{\mathcal{H}}$ – the map between $\Phi(x)$ and $\hat{f}(x)$ is one-to-one.

Assuming our input x is a discrete random variable, $\Phi(x)$ and $\hat{f}(x)$ are also discrete random variables. The one-to-one relationship then is sufficient to prove that mutual information between two random variables is 0 (i.e. $I(\Phi(x); \hat{f}(x)) = 0$). □

G.5 Accuracy of approximation methods

In the main text, we proposed three main measures of features: quality Q^*, utility $\hat{Q}$, and eigenvalues (or intensity) ρ. These measures require the true input distribution q, which we cannot access in practice. The approximation method for these measures was described in Appendix G.1 – the Nyström method for approximating e_k and the Riemann sum for the inner product in q. In this section, we use empirical methods - where the true eigenfunctions and eigenvalues are known- to assess the validity of Nyström method in approximating eigenvalues and eigenfunctions. We also discuss why quality and utility were presented in a cumulative manner (Π^* and $\hat{\Pi}$) while the eigenvalues (ρ_k) are presented directly.

G.5.1 The Nyström method

As discussed in Appendix G.1, the Nyström method diagonalizes the empirical feature matrix $\Phi(X) \in \mathbb{R}^{p\times n}$ to approximate the eigenfunctions e_k and eigenvalues ρ_k. This is more explicitly written in lines 6 and 7 of Algorithm 12:

$$\rho_k \leftarrow s_k^2, \qquad e_k(x) \leftarrow u_k^T \Phi(x)/s_k, \tag{G.13}$$

where u and s are the left orthogonal matrix and singular values obtained from singular value decomposition of $\Phi(X)$.

Experiment setup for testing Nyström method

The accuracy of the Nyström method depends on the accuracy of singular values (s_k) and left eigenvectors (u_k) of a random matrix ($\Phi(X)$). To empirically assess the accuracy of the method, we will use p independent Gaussian random variables to sample our feature matrix. A power-law distribution with exponent α will be used as the eigenvalues:

$$\Phi_k(x) \sim \sqrt{\rho_k}\mathcal{N}(0,1), \qquad \rho_k = k^{-\alpha}. \tag{G.14}$$

Gaussian random variables serve as valid features since the features are random variables - $x \sim q$ is a random variable, and e_k is a fixed map, thus $e_k(x)$, $x \sim q$ is

also a random variable - and are orthogonal to other eigenfunctions:

$$e_k(x) : Z_k \sim \mathcal{N}(0,1), \qquad \langle e_i | e_j \rangle = \frac{1}{2\pi} \int_{Z_i, Z_j} Z_i Z_j e^{-\frac{1}{2}(Z_i^2 + Z_j^2)} dZ_i dZ_j = \delta_{ij}. \quad \text{(G.15)}$$

Eigenfunctions can also be expressed with Φ as

$$e_k(x) = o_k^T \Phi(x) / \sqrt{\rho_k} \quad \text{(G.16)}$$

where $o_k \in \mathbf{p}$ is one-hot vector with its k^{th} entry equal to 1 and all other entries 0 (i.e. $o_k = [0, \cdots, 0, 1, 0, \cdots, 0]$). Indeed we can check that e_k is the eigenfunction of T with eigenvalue ρ_k

$$T[\frac{o_k^T \Phi(x)}{\sqrt{\rho_k}}] = \int \Phi(x')^T \Phi(x) \frac{o_k^T \Phi(x)}{\sqrt{\rho_k}} q(x) dx \quad \text{(G.17)}$$

$$= \Phi(x')^T \int Z_k^2 \sqrt{\frac{\rho_k}{2\pi}} e^{-\frac{Z_k^2}{2}} dZ_k \quad \text{(G.18)}$$

$$= \sqrt{\rho_k} \Phi(x')^T o_k \quad \text{(G.19)}$$

$$= \rho_k e_k, \quad \text{(G.20)}$$

where in the second line, we used that o_k^T is a one-hot vector.

Accuracy measures

With the underlying true features and eigenvalues defined, we propose measures for calculating the accuracy of approximated values obtained via Algorithm 12.

The error of eigenvalue is measured as square distance between ρ_k (true value) and $s_k^2(X)$ (approximation) normalized by ρ_k:

$$\mathrm{E}_X \left[\frac{(s_k^2(X) - \rho_k)^2}{\rho_k^2} \right], \quad \text{(G.21)}$$

Note that $\mathrm{E}_X[]$ means expectation over all possible features matrices $\Phi(X)$.

The error of eigenfunction is measured as 1 minus the inner product between $u_k(X) \in \mathbb{R}^p$ and o_k:

$$1 - \mathbf{E}_X \left[|u_k^T o_k| \right]. \quad \text{(G.22)}$$

The inner product $u_k^T o_k$ is proportional to the inner product between e_k^{approx} and e_k:

$$\langle e_k^{approx}, e_k \rangle = \int \left(\frac{u_k(X)^T \Phi(x)}{s_k(X)} \right)^T \frac{o_k^T \Phi(x)}{\sqrt{\rho_k}} q(x) dx \tag{G.23}$$

$$= \frac{u_k(X)^T}{s_k(X)} \int \Phi(x)^T \Phi(x) q(x) dx \frac{o_k}{\sqrt{\rho_k}} \tag{G.24}$$

$$= \frac{u_k(X)^T}{s_k(X)} Diag(\rho_k) \frac{o_k}{\sqrt{\rho_k}} \tag{G.25}$$

$$= \frac{u_k(X)^T o_k}{s_k(X)\sqrt{\rho_k}}, \tag{G.26}$$

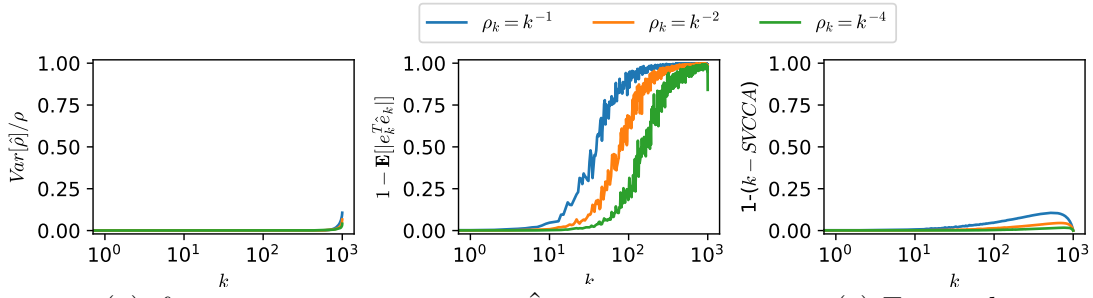
where the denominator is not of interest as we are not particularly interested in the norm of the e_k^{approx}.

Results

In Fig. G.4 we generate a feature matrix containing $n = 10,000$ datapoints and $p = 1000$ features with Gaussian random variables. We use the Nyström method to approximate the eigenvalues and eigenfunctions and assess their accuracy with the metrics in Eq. (G.21) and Eq. (G.22). We observe in Fig. G.4(a) that eigenvalues are approximated precisely except for eigenvalues with $k \approx p$. In Fig. G.4(b), we observe that the accuracy of the eigenfunctions correlates with the respective eigenvalues: the larger the eigenvalue (compared to the median eigenvalue), the better the accuracy. In the main text, we are often interested only in the first few eigenfunctions with large eigenvalues; We can conclude that the Nyström method is sufficiently accurate for our needs.

G.5.2 Function space approximation

The Nyström method's prediction accuracy degrades for smaller eigenvalues. The spanned space of $[e_1^{approx}, \cdots, e_k^{approx}]$, however, often accurately approximates that of $[e_1, \cdots, e_k]$; The larger error of the e_k^{approx} is often due to the cumulation of error in predicting $e_{i<k}^{approx}$. Fig. G.5 illustrates an example where $[u_1, u_2, u_3]$ span the same space as $[o_1, o_2, o_3]$, and u_1 and u_2 (thus e_1 and e_2) are well approximated – the errors in predicting u_1 and u_2 accumulate to the error of predicting u_3, and results in a poorer approximation for u_3.



(a) f^* projection (b) $\hat{f}$ projection (c) Eigenvalues

Figure G.4: Nyström approximation for different eigenvalue distributions A feature matrix was sampled from Gaussian random variables and then approximated with Nyström method. Power law eigenvalue spectra with different decaying exponents were tested. (a) The eigenvalues are accurately predicted except for the tail eigenvalues with $k \approx p$. (b) The eigenfunctions are accurate only when the eigenvalues are large compared to the median eigenvalue. (c) The function space error below 0.25 for all k.

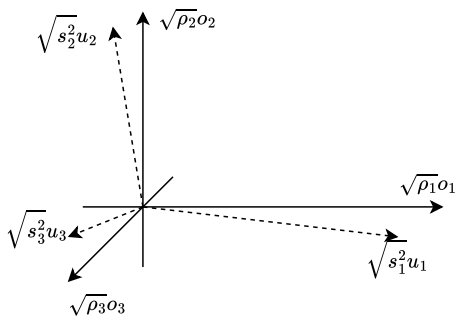


Figure G.5: Approximation on different eigenvalue distributions ...

We can use the Gaussian random variable setup to check the error of the function space approximation. To measure the similarity between $span([e_1, \cdots, e_k])$ and approximated function space $[e_1^{approx}, \cdots, e_k^{approx}]$, we can use k-SVCCA [123] or the cumulative projection measure in Eq. (6.10).

$$\frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{k} \langle e_i | e_j^{approx} \rangle^2 . \tag{G.27}$$

Since we can express the inner product between true and approximated eigenfunctions with u_k and o_k (Eq. (G.23)), we can express the metric in terms of u_k and o_k:

$$1 - \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{k} \left(u_i^T o_j \right)^2 , \tag{G.28}$$

where we subtracted the measure from 1 so the measure will be 0 when there is no error. In Fig. G.4(c), we observe that the function space can be approximated

more accurately in comparison to individual eigenfunctions in Fig. G.4(b); This is why we often present the quality and utility in cumulative measure Π^* or $\hat{\Pi}$.

G.5.3 Error in projection measures

We now discuss the error in the projection measures like Π^*. The projection measure (Eq. (6.8)) consists of the sum of squares of inner products between eigenfunctions and function of interest (target or expressed). The Riemann sum over the **test** set approximates the inner product – similar to how test loss approximates generalization loss.

$$\frac{\langle e_k, f_j\rangle^2}{\|e_k\|_2^2\|f_j\|_2^2} = \frac{(\int_\chi e_k(x)f_j(x)q(x)dx)^2}{\int_\chi e_k(x)^2q(x)dx\int_\chi f_j(x)^2q(x)dx} \approx \frac{\left(\sum_{x^{(i)}\in S_{te}} e_k(x^{(i)})f_j(x^{(i)})\right)^2}{\sum_{x^{(i)}\in S_{te}} e_k(x^{(i)})^2\sum_{x^{(i)}\in S_{te}} f_j(x^{(i)})^2}. \tag{G.29}$$

Using the Riemann sum, we can measure the correlation between spanned spaces (Eq. (6.10)). Using the vector representation of eigenfunction in the test set $e_i^{approx}(X_{test}) \in \mathbb{R}^{n_{test}}$ and the vector representation of function of interest $f_i(X_{test})$, we can approximate Eq. (6.10) as

$$\sum_{j=1}^{C}\sum_{i=1}^{p}\langle e_i|f_j\rangle^2 \approx \mathrm{Tr}(EF^TFE^T) \qquad , E_{ij} = \frac{e_i^{approx}(x^{(j)})}{\sum_j^n E_{ij}^2}, \qquad F_{ij} = \frac{f_i(x^{(j)})}{\sum_j^n F_{ij}^2}. \tag{G.30}$$

where $x^{(j)}$ are iterated over the test set. If the rows of $E \in \mathbb{R}^{p\times n_{test}}$ and $F \in \mathbb{R}^{C\times n_{test}}$ are orthogonal, then Eq. (G.30) is smaller than equal to 1: consistent with the definition of Eq. (6.10).

However, Eq. (G.30) is susceptible to error when: rows of E are not orthogonal and when $p \approx n_{test}$. We will discuss how we handled such problems in the paper.

Non orthogonal eigenfunctions

As discussed earlier, approximated eigenfunctions are less accurate when their eigenvalues are smaller; they may not even be orthogonal in q or the empirical distribution

of the test set. The inner product between two approximated eigenfunctions is

$$\langle e_i^{approx} | e_j^{approx} \rangle = \frac{1}{s_i s_j} \int_{\mathcal{X}} u_i^T \Phi(x) \Phi^T(x) u_j q(x) dx \tag{G.31}$$

$$= \frac{1}{s_i s_j} u_i^T \left(\int_{\mathcal{X}} \Phi(x) \Phi^T(x) q(x) dx \right) u_j \tag{G.32}$$

$$= \frac{1}{s_i s_j} u_i^T J u_j, \tag{G.33}$$

where the Fisher information matrix $J \in \mathbb{R}^{p \times p}$ [270] is defined as $\int_{\mathcal{X}} \Phi(x) \Phi^T(x) q(x) dx$. Eq. (G.31) equals Kronecker delta if only if $[u_1, \cdots, u_p]$ are the eigenvectors of J with eigenvalues $[s_1^2, \cdots, s_p^2]$. Since we obtained u_i from the empirical feature matrix of the training set, the approximated eigenfunctions, especially those with small eigenvalues, are often not orthogonal in q (and in the test set).

To handle such an issue, we use QR decomposition on

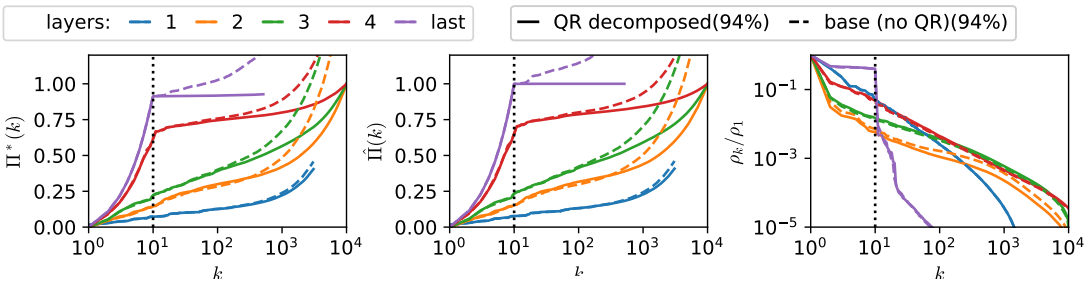
$$E = [e_1^{approx}(X_{test}), \cdots, e_p^{approx}(X_{test})] \in \mathbb{R}^{p \times n_{test}}$$

before applying Eq. (G.30): because QR decomposition orthogonalizes the rows; preserves the span of the first k vectors; handles overlapping vector between rows by conserving larger eigenvalue eigenvectors (eigenfunctions) - which are more accurate.

In Fig. G.6, we plot the cumulative measure with and without QR decomposition for ResNet18 on CIFAR10. We observe that Π^* and $\hat{\Pi}$ grow beyond 1 without QR decomposition, indicating that approximated eigenfunctions are not orthogonal; The inconsistent behaviour is removed when QR decomposition is applied. Note that the deviation between QR and no-QR plots emerges when the eigenvalues are sufficiently small (larger error in approximating the eigenfunction).

Finite size effect

For two sets of basis functions, increasing the number of bases to a large but finite value does not guarantee any overlap between the two function spaces: because functions are infinite-dimensional vectors. This is not the case for two finite-dimensional matrices: If $p = n_{test}$ in Eq. (G.30) and E has full rank, then any F will be spanned by the rows of E.



(a) f^* projection (b) $\hat{f}$ projection (c) Eigenvalues

Figure G.6: Comparing approximation methods for the projection measures ResNet18 on CIFAR10. For (a) and (b), the measures are calculated with – solid – and without – dashed – QR-composition. Note that QR decomposition removes the inconsistent behaviour of the cumulative projection measures (e.g. larger than 1). With QR decomposition, the projection measure for layers with $p \geq 10^4$ consistently equals 1 when $k = n_{test} = 10^4$.

In Fig. G.6(a,b), we observe that $\Pi^*(n_{test})$ and $\hat{\Pi}(n_{test})$ are equal to 1 for layers with $p > n_{test}$. This does not indicate that intermediate features can express the target/expressed function; It is achieved because n_{test} orthogonal vectors (E) can always span n_{test} dimensional vector space (and thus the row space of F).

G.6 The first eigenfunction of the T_{MP} operator is a constant

MP-operators (Definition 12) include a constant function in addition to a more conventional projection operator onto $\hat{\mathcal{H}}$, the C-dimensional function space spanned by entries of the learned function $\hat{f} : \mathcal{X} \to \mathbb{R}^C$. This difference with simple projection operators means that an MP-operator uniformly rescales all function components in $\hat{\mathcal{H}}$ in an isotropic manner, with the exception of the direction of the constant function. The fact that the constant function exists in the feature kernel's reproducing kernel Hilbert space (RKHS) or simply the function space is not surprising as the final output function space typically includes the constant function.[1] More importantly, a bias term in the last layer on a DNN adds a constant function to the entries of the output, which necessitates the existence of a constant function in the RKHS of its kernel.

[1] Unless it is centered to 0.

What is perhaps more surprising is that the constant function is the first eigenfunction of T_{MP} with the largest eigenvalue. Interestingly, this function is often disregarded in other analyses, for example in principal component analysis (PCA) and in measurements of CKA. We also observe that the first eigenfunction of T often closely resembles the constant function (even when not at its limit T_{MP}). Across all our experiments, except for the 1-D toy model in Fig. 6.4, we consistently observe $\langle e_1|\mathbf{1}\rangle > 0.95$. Due to this observation, we handle the first eigenvalue ρ_1 as an exception, for example when calculating $D_{eff}(\rho)$ or defining the MF regime.

We also find that at initialisation, the first eigenfunction tends to dominate the initialised function (i.e. $\Pi\hat{(}1) \approx 1$) for architectures such as ResNet18 and VGG16. Moreover, these architectures have $\langle e_1|\mathbf{1}\rangle \approx 1$ even at initialization. One possible explanation for this behaviour is simplicity bias [65, 64, 1, 5], which predicts that simple (low Kolmogorov complexity) outputs (functions) are more exponentially likely to appear if the input-output map is simple enough. However, the exact reason why the constant function is the first eigenfunction of the feature kernel requires further study. For the purposes of this paper, we accept this observation as a given.

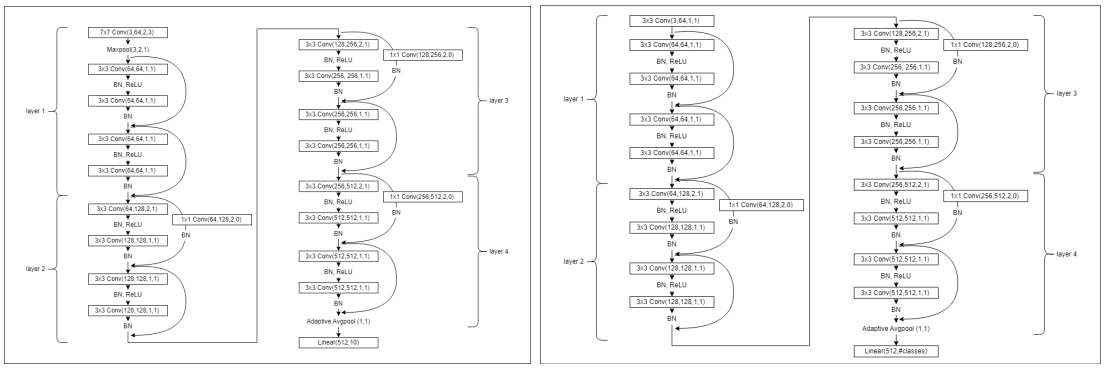
The constant function also helps explain why the simplex equiangular tight frame (ETF) emerges in NC as the ideal max-margin feature for classification, instead of an orthonormal basis: A $(C-1)$-dimensional simplex ETF can be obtained as the projection of the standard basis onto the orthogonal complement of $[\frac{1}{C}, \frac{1}{C}, \cdots, \frac{1}{C}]$, which is the average vector of all basis vectors, inside $\mathbb{R}^C$. For example, projecting $[0,0,1]$, $[0,1,0]$, and $[1,0,0]$ onto the orthogonal complement of $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$ gives three vertices $[-\frac{1}{3}, -\frac{1}{3}, \frac{2}{3}]$, $[-\frac{1}{3}, \frac{2}{3}, -\frac{1}{3}]$, $[\frac{2}{3}, -\frac{1}{3}, -\frac{2}{3}]$ which forms an (ordinary plane) equilateral triangle.

G.7 Experimental details

G.7.1 Architecture details

$k \times k$ Conv$(\cdot,\cdot,\cdot,\cdot)$ refers to the 2d convolutional module with a kernel of size kxk, and arguments within parenthesis refer to the input channel, output channel, stride,

and padding respectively. Linear$(\cdot,\cdot)$ refers to the fully connected module with the arguments referring to input width and output width respectively. MaxPool$(\cdot,\cdot,\cdot)$ refers to max pooling with the arguments as kernel size, stride, and padding respectively. Adaptive Avgpool$(\cdot,\cdot)$ refers to adaptive average pooling where arguments are the size of the image after pooling. The curved arrow refers to the skip connection (identity), and any modules that appear in the arrow are applied. BN refers to batch normalization and (BN, ReLU) refers to the application of batch normalization and ReLU in respective order. The definition of the term "layer" for each model is also provided.

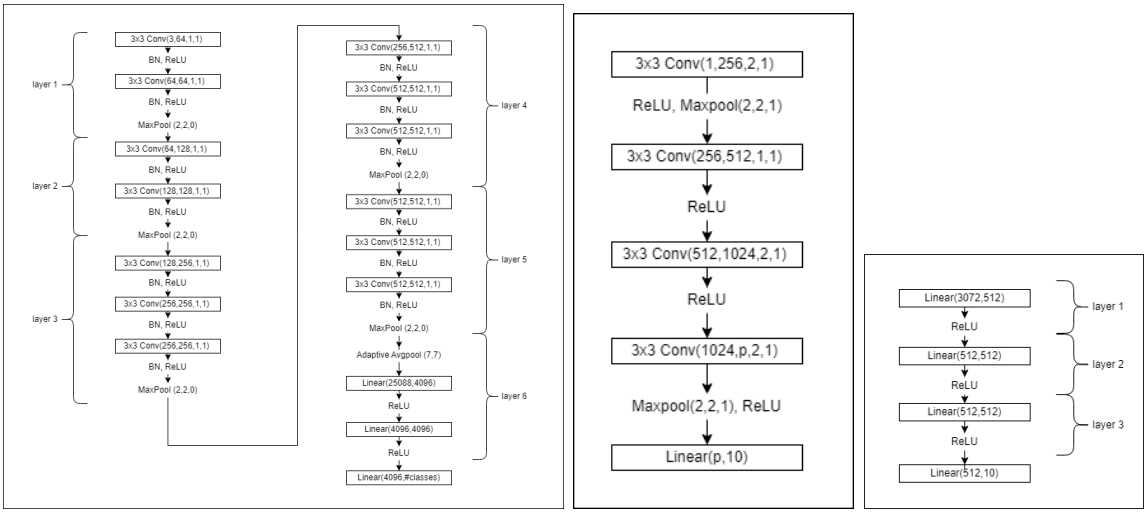


(a) ResNet with max pooling **(b)** ResNet without max pooling

Figure G.7: ResNet18 architectures (a): This is the ResNet proposed in [118]. The size of feature space(post-activation) for each layer is 4096, 2048, 1024, and 512 respectively. When the initial stride is decreased to 1 (i.e. 7x7 conv(3,64,**1**,3) for the initial convolution module), the feature space size increases to 16832, 8192, 4096, 512 respectively. **(b):** The architecture is similar to (a) but lacks a max pooling module and has a smaller stride in the initial convolution module; The feature space is larger with the size of 65536, 32768, 16384, and 512.

G.7.2 Learning rate and hyperparameters

In our experiments, unless explicitly stated, SGD with a momentum of 0.9 and weight decay of 0.001 was used. Adam [37] was used when the model did not converge with SGD. All models were trained for 200, 400, or 600 epochs: the earliest epoch that achieves above 99% training accuracy. For models trained for 200 epochs, a scheduler with 0.2 decay for every 60 epochs was used. For models that require



(a) VGG16 **(b)** CNN **(c)** FCN

Figure G.8: VGG16, CNN, and FCN architectures (a): VGG16 The model has been segmented into layers by maxpooling. The feature space size for each layer is 16384, 8192, 4096, 2048, 512, and 4096. Note that the penultimate layer only consists of linear modules (fully connected) and non-linearities.

400 or more epochs, the scheduler was removed. If the model did not converge in 600 epochs or achieve below 99% training accuracy, it would be explicitly mentioned.

CIFAR10 and CIFAR100 were normalized to the mean of (0.4914, 0.4822, 0.4465) and standard deviation of (0.2023, 0.1994, 0.2010). For data augmentation, a random crop of 32 with padding of 4 and a random horizontal flip was used. No augmentation or whitening was applied to the MNIST dataset.

G.7.3 Loss

We use MSE loss with a centered one-hot vector (i.e. $\sum_i^c f_i^*(x) = 0$ for all $x \in \mathcal{X}$) as the C dimensional target function $f^* : \mathcal{X} \to \mathbb{R}^C$. Before centering, we have scaled the f^* by a factor of 3 for CIFAR10 and MNIST; and by 10 for CIFAR100.

H

Appendices for Chapter 7

H.1 Proofs

Here are the proofs for the theoretical results in the main text.

Proposition 7.2.1 (Decomposition of linearisation error)**.** *For any differentiable loss ℓ and any differentiable machine learning model $\boldsymbol{f}$ the linearisation error of the objective function $\mathcal{L}$ admits the following decomposition:*

$$\underbrace{\Delta\mathcal{L}(\boldsymbol{w}) - \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top \Delta\boldsymbol{w}}_{\textit{linearisation error of objective}} = \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top \underbrace{[\Delta\boldsymbol{f}(\boldsymbol{x}) - \nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x})\Delta\boldsymbol{w}]}_{\textit{linearisation error of model}} + \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \underbrace{\Delta\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y}) - \nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top\Delta\boldsymbol{f}(\boldsymbol{x})}_{\textit{linearisation error of loss}}.$$

Proof. By the chain rule, $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top\Delta\boldsymbol{w} = \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top\nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x})\Delta\boldsymbol{w}$. Therefore:

$$\Delta\mathcal{L}(\boldsymbol{w}) - \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top\Delta\boldsymbol{w} = \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \Delta\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y}) - \nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top\nabla_{\boldsymbol{w}}\boldsymbol{f}(\boldsymbol{x})\Delta\boldsymbol{w}.$$

Adding and subtracting $\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}} \nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top\Delta\boldsymbol{f}(\boldsymbol{x})$ on the right-hand side yields the result. □

Lemma 1 (Bregman divergence of square loss)**.** *When ℓ is set to square loss, then:*

$$\text{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{f}(\boldsymbol{x}), \Delta\boldsymbol{f}(\boldsymbol{x})) = \tfrac{1}{2d_L}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_2^2.$$

Proof. Expanding the Euclidean norms in the loss perturbation $\Delta\ell$ yields:

$$\begin{aligned}\Delta\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y}) &= \tfrac{1}{2d_L}\|\boldsymbol{f}(\boldsymbol{x})+\Delta\boldsymbol{f}(\boldsymbol{x})-\boldsymbol{y}\|_2^2 - \tfrac{1}{2d_L}\|\boldsymbol{f}(\boldsymbol{x})-\boldsymbol{y}\|_2^2 \\ &= \tfrac{1}{2d_L}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_2^2 + (\boldsymbol{f}(\boldsymbol{x})-\boldsymbol{y})^\top\Delta\boldsymbol{f}(\boldsymbol{x}).\end{aligned}$$

The result follows by identifying that $\nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top\Delta\boldsymbol{f}(\boldsymbol{x}) = (\boldsymbol{f}(\boldsymbol{x})-\boldsymbol{y})^\top\Delta\boldsymbol{f}(\boldsymbol{x})$.

□

Lemma 2 (Bregman divergence of xent loss)**.** *When ℓ is set to cross-entropy loss, and if $\boldsymbol{y}^\top\mathbf{1}=1$, then:*

$$\begin{aligned}\operatorname{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{f}(\boldsymbol{x}),\Delta\boldsymbol{f}(\boldsymbol{x})) &= D_{\mathrm{KL}}\Big(\operatorname{softmax}(\boldsymbol{f}(\boldsymbol{x}))\,\Big\|\,\operatorname{softmax}(\boldsymbol{f}(\boldsymbol{x})+\Delta\boldsymbol{f}(\boldsymbol{x}))\Big) \\ &\le \tfrac{1}{2}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_\infty^2 + \mathcal{O}(\Delta\boldsymbol{f}^3).\end{aligned}$$

Proof. First, since $\sum_i \boldsymbol{y}_i = 1$, cross-entropy loss may be re-written:

$$\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y}) := -\log[\operatorname{softmax}(\boldsymbol{f}(\boldsymbol{x}))]^\top\boldsymbol{y} = -\boldsymbol{f}(\boldsymbol{x})^\top\boldsymbol{y} + \log\|\exp\boldsymbol{f}(\boldsymbol{x})\|_1.$$

The linear term $-\boldsymbol{f}(\boldsymbol{x})^\top\boldsymbol{y}$ does not contribute to the linearisation error and may be neglected. Therefore:

$$\begin{aligned}&\Delta\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y}) - \nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top\Delta\boldsymbol{f}(\boldsymbol{x}) \\ &\quad= \log\|\exp(\boldsymbol{f}(\boldsymbol{x})+\Delta\boldsymbol{f}(\boldsymbol{x}))\|_1 - \log\|\exp\boldsymbol{f}(\boldsymbol{x})\|_1 - \nabla_{\boldsymbol{f}(\boldsymbol{x})}\log\|\exp\boldsymbol{f}(\boldsymbol{x})\|_1^\top\Delta\boldsymbol{f}(\boldsymbol{x}) \\ &\quad= \log\frac{1/\|\exp\boldsymbol{f}(\boldsymbol{x})\|_1}{1/\|\exp(\boldsymbol{f}(\boldsymbol{x})+\Delta\boldsymbol{f}(\boldsymbol{x}))\|_1} - \frac{\exp\boldsymbol{f}(\boldsymbol{x})^\top}{\|\exp\boldsymbol{f}(\boldsymbol{x})\|_1}\Delta\boldsymbol{f}(\boldsymbol{x}) \\ &\quad= \frac{\exp\boldsymbol{f}(\boldsymbol{x})^\top}{\|\exp\boldsymbol{f}(\boldsymbol{x})\|_1}\log\frac{\exp\boldsymbol{f}(\boldsymbol{x})/\|\exp\boldsymbol{f}(\boldsymbol{x})\|_1}{\exp(\boldsymbol{f}(\boldsymbol{x})+\Delta\boldsymbol{f}(\boldsymbol{x}))/\|\exp(\boldsymbol{f}(\boldsymbol{x})+\Delta\boldsymbol{f}(\boldsymbol{x}))\|_1}.\end{aligned}$$

The final line is equivalent to $D_{\mathrm{KL}}\Big(\operatorname{softmax}(\boldsymbol{f}(\boldsymbol{x}))\,\Big\|\,\operatorname{softmax}(\boldsymbol{f}(\boldsymbol{x})+\Delta\boldsymbol{f}(\boldsymbol{x}))\Big)$ establishing the first equality.

To establish the inequality, let $\otimes$ denote the outer product and define $p :=$

softmax($f(\boldsymbol{x})$). Then we have:

$$\begin{aligned}\Delta\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y}) - \nabla_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})^\top\Delta\boldsymbol{f}(\boldsymbol{x}) &= \frac{1}{2}\Delta\boldsymbol{f}(\boldsymbol{x})^\top\nabla^2_{\boldsymbol{f}(\boldsymbol{x})}\ell(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y})\Delta\boldsymbol{f}(\boldsymbol{x}) + \mathcal{O}(\Delta\boldsymbol{f}^3)\\ &= \frac{1}{2}\Delta\boldsymbol{f}(\boldsymbol{x})^\top\nabla^2_{\boldsymbol{f}(\boldsymbol{x})}\log\|\exp\boldsymbol{f}(\boldsymbol{x})\|_1\Delta\boldsymbol{f}(\boldsymbol{x}) + \mathcal{O}(\Delta\boldsymbol{f}^3)\\ &= \frac{1}{2}\Delta\boldsymbol{f}(\boldsymbol{x})^\top[\operatorname{diag}(p) - p\otimes p]\Delta\boldsymbol{f}(\boldsymbol{x}) + \mathcal{O}(\Delta\boldsymbol{f}^3)\\ &\leq \frac{1}{2}\Delta\boldsymbol{f}(\boldsymbol{x})^\top\operatorname{diag}(p)\Delta\boldsymbol{f}(\boldsymbol{x}) + \mathcal{O}(\Delta\boldsymbol{f}^3)\\ &\leq \frac{1}{2}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|^2_\infty + \mathcal{O}(\Delta\boldsymbol{f}^3),\end{aligned}$$

where we have used that $p\otimes p$ is positive definite and then applied Hölder's inequality with $\|p\|_1 = 1$. □

Theorem 4 (Functional expansion)**.** *Consider a convex differentiable loss ℓ and a differentiable machine learning model $\boldsymbol{f}$. Under Assumption 1, the corresponding composite objective $\mathcal{L}$ admits the expansion:*

$$\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) = \underbrace{\mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top\Delta\boldsymbol{w}}_{\textit{first-order Taylor series}} + \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\operatorname{bregman}_{\ell(\cdot,\boldsymbol{y})}(\boldsymbol{f}(\boldsymbol{x}),\Delta\boldsymbol{f}(\boldsymbol{x})).$$

Proof. The result follows by substituting Assumption 1 into Proposition 7.2.1 and applying Definition 22. □

Corollary 7.2.1 (Functional expansion of mean squared error)**.** *Under Assumption 1, for square loss:*

$$\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top\Delta\boldsymbol{w} + \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\tfrac{1}{2d_L}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|^2_2.$$

Proof. Combine Lemma 1 with Theorem 4 to obtain the result. □

Corollary 7.2.2 (Functional majorisation for xent loss)**.** *Under Assumption 1, for cross-entropy loss, if $\boldsymbol{y}^\top\mathbf{1} = 1$:*

$$\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) \leq \mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top\Delta\boldsymbol{w} + \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\tfrac{1}{2}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|^2_\infty + \mathcal{O}(\Delta\boldsymbol{f}^3).$$

Proof. Combine Lemma 2 with Theorem 4 to obtain the result. □

Lemma 3 (Output bound)**.** *The output norm of a fully-connected network* $\boldsymbol{f}$ *obeys the following bound:*

$$\|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})\|_2 \leq \left[\prod_{k=1}^{L} \|\boldsymbol{W}_k\|_*\right] \times \|\boldsymbol{x}\|_2 = \sqrt{d_L} \text{ under Prescription 1.}$$

Proof. For any vector $\boldsymbol{v}$ and matrix $\boldsymbol{M}$ with compatible dimensions, we have that $\|\boldsymbol{M}\boldsymbol{v}\|_2 \leq \|\boldsymbol{M}\|_* \cdot \|\boldsymbol{v}\|_2$ and $\|\operatorname{relu} \boldsymbol{v}\|_2 \leq \|\boldsymbol{v}\|_2$. The lemma follows by applying these results recursively over the depth of the network. □

Lemma 4 (Deep relative trust)**.** *When adjusting the weights* $\boldsymbol{w} = (\boldsymbol{W}_1, ..., \boldsymbol{W}_L)$ *of a fully-connected network* $\boldsymbol{f}$ *by* $\Delta\boldsymbol{w} = (\Delta\boldsymbol{W}_1, ..., \Delta\boldsymbol{W}_L)$*, the induced functional perturbation* $\Delta\boldsymbol{f}(\boldsymbol{x}) := \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w} + \Delta\boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w})$ *obeys:*

$$\begin{aligned}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_2 &\leq \left[\prod_{k=1}^{L} \|\boldsymbol{W}_k\|_*\right] \times \|\boldsymbol{x}\|_2 \times \left[\prod_{k=1}^{L} \left(1 + \frac{\|\Delta\boldsymbol{W}_k\|_*}{\|\boldsymbol{W}_k\|_*}\right) - 1\right] \\ &\leq \sqrt{d_L} \times (\exp \eta - 1) \text{ under Prescription 1.}\end{aligned}$$

Proof. We proceed by induction. First, consider a network with $L = 1$ layers: $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}_1\boldsymbol{x}$. Observe that $\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_2 = \|\Delta\boldsymbol{W}_1\boldsymbol{x}\|_2 \leq \|\Delta\boldsymbol{W}_1\|_* \cdot \|\boldsymbol{x}\|_2$ as required. Next, assume that the result holds for a network $\boldsymbol{g}(\boldsymbol{x})$ with $L-1$ layers and consider adding a layer to obtain $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}_L \circ \operatorname{relu} \circ \boldsymbol{g}(\boldsymbol{x})$. Then:

$$\begin{aligned}\|\Delta\boldsymbol{f}(\boldsymbol{x})\|_2 &= \|(\boldsymbol{W}_L + \Delta\boldsymbol{W}_L) \circ \operatorname{relu} \circ (\boldsymbol{g}(\boldsymbol{x}) + \Delta\boldsymbol{g}(\boldsymbol{x})) - \boldsymbol{W}_L \circ \operatorname{relu} \circ \boldsymbol{g}(\boldsymbol{x})\|_2 \\ &= \|\boldsymbol{W}_L (\operatorname{relu} \circ (\boldsymbol{g}(\boldsymbol{x}) + \Delta\boldsymbol{g}(\boldsymbol{x})) - \operatorname{relu} \circ \boldsymbol{g}(\boldsymbol{x})) \\ &\qquad + \Delta\boldsymbol{W}_L (\operatorname{relu} \circ (\boldsymbol{g}(\boldsymbol{x}) + \Delta\boldsymbol{g}(\boldsymbol{x})) - \operatorname{relu}(0)) \|_2 \\ &\leq \|\boldsymbol{W}_L\|_* \cdot \|\Delta\boldsymbol{g}(\boldsymbol{x})\|_2 + \|\Delta\boldsymbol{W}_L\|_* \cdot (\|\boldsymbol{g}(\boldsymbol{x})\|_2 + \|\Delta\boldsymbol{g}(\boldsymbol{x})\|_2) \\ &= (\|\boldsymbol{W}_L\|_* + \|\Delta\boldsymbol{W}_L\|_*) \cdot \|\Delta\boldsymbol{g}(\boldsymbol{x})\|_2 + \|\Delta\boldsymbol{W}_L\|_* \cdot \|\boldsymbol{g}(\boldsymbol{x})\|_2,\end{aligned}$$

where the inequality follows by applying the triangle inequality, the operator norm bound, the fact that relu is one-Lipschitz, and a further application of the triangle inequality. But by the inductive hypothesis and Lemma 3, the right-hand side is

bounded by:

$$
\begin{aligned}
&(\|\boldsymbol{W}_L\|_* + \|\Delta\boldsymbol{W}_L\|_*)\left[\prod_{k=1}^{L-1}\left(1+\frac{\|\Delta\boldsymbol{W}_k\|_*}{\|\boldsymbol{W}_k\|_*}\right)-1\right]\times \\
&\left[\prod_{k=1}^{L-1}\|\boldsymbol{W}_k\|_*\right]\times\|\boldsymbol{x}\|_2 + \|\Delta\boldsymbol{W}_L\|_*\times\left[\prod_{k=1}^{L-1}\|\boldsymbol{W}_k\|_*\right]\times\|\boldsymbol{x}\|_2 \\
&=\left[\prod_{k=1}^{L}\left(1+\frac{\|\Delta\boldsymbol{W}_k\|_*}{\|\boldsymbol{W}_k\|_*}\right)-1\right]\times\left[\prod_{k=1}^{L}\|\boldsymbol{W}_k\|_*\right]\times\|\boldsymbol{x}\|_2.
\end{aligned}
$$

The induction is complete. To further bound this result under Prescription 1, observe that the product $\left[\prod_{k=1}^{L}\|\boldsymbol{W}_k\|_*\right]\times\|\boldsymbol{x}\|_2$ telescopes to just $\sqrt{d_L}$, while the other product satisfies:

$$
\left[\prod_{k=1}^{L}\left(1+\frac{\|\Delta\boldsymbol{W}_k\|_*}{\|\boldsymbol{W}_k\|_*}\right)-1\right]=\left(1+\frac{\eta}{L}\right)^L-1\leq\lim_{L\to\infty}\left(1+\frac{\eta}{L}\right)^L-1=\exp\eta-1.
$$

Combining these observations yields the result. □

Lemma 5 (Exponential majorisation)**.** *For an FCN with square loss, under Assumption 1 and Prescription 1:*

$$
\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w})\leq\mathcal{L}(\boldsymbol{w})+\frac{\eta}{L}\sum_{k=1}^{L}\left[\sqrt{d_k/d_{k-1}}\times\operatorname{tr}\frac{\Delta\boldsymbol{W}_k^\top\nabla_{\boldsymbol{W}_k}\mathcal{L}}{\|\Delta\boldsymbol{W}_k\|_*}\right]+\tfrac{1}{2}(\exp\eta-1)^2.
$$

Proof. Substitute Lemma 4 into Corollary 7.2.1 and decompose $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top\Delta\boldsymbol{w}=\sum_{k=1}^{L}\operatorname{tr}(\Delta\boldsymbol{W}_k^\top\nabla_{\boldsymbol{W}_k}\mathcal{L})$. The result follows by realising that under Prescription 1, the perturbations satisfy $\|\Delta\boldsymbol{W}_k\|_*=\sqrt{d_k/d_{k-1}}\cdot\frac{\eta}{L}$. □

Theorem 5 (Automatic gradient descent)**.** *For a deep fully-connected network, under Assumptions 1 and 2 and Prescription 1, the majorisation of square loss given in Lemma 5 is minimised by setting:*

$$
\eta=\log\frac{1+\sqrt{1+4G}}{2},\qquad\Delta\boldsymbol{W}_k=-\frac{\eta}{L}\cdot\sqrt{d_k/d_{k-1}}\cdot\frac{\nabla_{\boldsymbol{W}_k}\mathcal{L}}{\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F},\qquad\textit{for all layers } k=1,\ldots,L.
$$

Proof. The inner product $\operatorname{tr}\frac{\Delta\boldsymbol{W}_k^\top\nabla_{\boldsymbol{W}_k}\mathcal{L}}{\|\Delta\boldsymbol{W}_k\|_*}$ that appears in Lemma 5 is most negative when the perturbation $\Delta\boldsymbol{W}_k$ satisfies $\Delta\boldsymbol{W}_k/\|\Delta\boldsymbol{W}_k\|_*=-\nabla_{\boldsymbol{W}_k}\mathcal{L}/\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_*$. Substituting this result back into Lemma 5 yields:

$$
\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w})\leq\mathcal{L}(\boldsymbol{w})-\frac{\eta}{L}\sum_{k=1}^{L}\left[\sqrt{d_k/d_{k-1}}\times\frac{\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F^2}{\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_*}\right]+\tfrac{1}{2}(\exp\eta-1)^2.
$$

Under Assumption 2, we have that $\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F^2/\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_* = \|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F$ and so this inequality simplifies to:

$$\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) \le \mathcal{L}(\boldsymbol{w}) - \eta\cdot G + \tfrac{1}{2}(\exp\eta - 1)^2.$$

Taking the derivative of the right-hand side with respect to η and setting it to zero yields $(\exp\eta - 1)\exp\eta = G$. Applying the quadratic formula and retaining the positive solution yields $\exp\eta = \frac{1}{2}(1+\sqrt{1+4G})$. Combining this with the relation that $\Delta\boldsymbol{W}_k/\|\Delta\boldsymbol{W}_k\|_* = -\nabla_{\boldsymbol{W}_k}\mathcal{L}/\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_*$ and applying that $\|\Delta\boldsymbol{W}_k\|_* = \sqrt{d_k/d_{k-1}}\cdot\frac{\eta}{L}$ by Prescription 1 yields the result. □

Lemma 6 (Bounded objective)**.** *For square loss, the objective is bounded as follows:*

$$\mathcal{L}(\boldsymbol{w}) \le \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{\|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})\|_2^2+\|\boldsymbol{y}\|_2^2}{2d_L} \le 1 \text{ under Prescription 1.}$$

Proof. The result follows by the following chain of inequalities:

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}) &:= \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{1}{2d_L}\|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})-\boldsymbol{y}\|_2^2 \le \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{1}{2d_L}(\|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})\|_2^2+\|\boldsymbol{y}\|_2^2)\\ &\le \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{d_L+d_L}{2d_L} = 1,\end{aligned}$$

where the second inequality holds under Prescription 1. □

Lemma 7 (Bounded gradient)**.** *For square loss, the norm of the gradient at layer k is bounded as follows:*

$$\|\nabla_{\boldsymbol{W}_k}\mathcal{L}\|_F \le \frac{\prod_{l=1}^L\|\boldsymbol{W}_l\|_*}{\|\boldsymbol{W}_k\|_*}\cdot\sqrt{\frac{2\mathcal{L}(\boldsymbol{w})}{d_L}}\cdot\sqrt{\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\|\boldsymbol{x}\|_2^2} \le \sqrt{2\cdot\frac{d_{k-1}}{d_k}}$$

under Prescription 1.

Proof. By the chain rule, the gradient of mean square error objective may be written:

$$\begin{aligned}\nabla_{\boldsymbol{W}_k}\mathcal{L}(\boldsymbol{w}) = \frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{1}{d_L}(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})-\boldsymbol{y})^\top\boldsymbol{W}_L\cdot\boldsymbol{D}_{L-1}\boldsymbol{W}_{L-1}\dots\\ \dots\boldsymbol{D}_{k+1}\boldsymbol{W}_{k+1}\cdot\boldsymbol{D}_k\otimes\boldsymbol{D}_{k-1}\boldsymbol{W}_{k-1}\dots\boldsymbol{D}_1\boldsymbol{W}_1\boldsymbol{x},\end{aligned}$$

where $\otimes$ denotes the outer product and $\boldsymbol{D}_k$ denotes a diagonal matrix whose entries are one when relu is active and zero when relu is inactive. Since the operator norm $\|\boldsymbol{D}_k\|_* = 1$, we have that the Frobenius norm $\|\nabla_{\boldsymbol{W}_k}\mathcal{L}(\boldsymbol{w})\|_F$ is bounded from above by:

$$
\begin{aligned}
&\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{1}{d_L}\Big\|(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})-\boldsymbol{y})^\top\boldsymbol{W}_L\cdot\boldsymbol{D}_{L-1}\boldsymbol{W}_{L-1}\ldots\boldsymbol{D}_{k+1}\boldsymbol{W}_{k+1}\cdot\boldsymbol{D}_k\otimes\boldsymbol{D}_{k-1}\boldsymbol{W}_{k-1}\ldots\boldsymbol{D}_1\boldsymbol{W}_1\boldsymbol{x}\Big\|_F\\
&=\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{1}{d_L}\Big\|(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})-\boldsymbol{y})^\top\boldsymbol{W}_L\cdot\boldsymbol{D}_{L-1}\boldsymbol{W}_{L-1}\ldots\boldsymbol{D}_{k+1}\boldsymbol{W}_{k+1}\cdot\boldsymbol{D}_k\Big\|_2\cdot\|\boldsymbol{D}_{k-1}\boldsymbol{W}_{k-1}\ldots\boldsymbol{D}_1\boldsymbol{W}_1\boldsymbol{x}\|_2\\
&\le\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{1}{d_L}\|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})-\boldsymbol{y}\|_2\cdot\|\boldsymbol{W}_L\|_*\cdot\|\boldsymbol{W}_{L-1}\|_*\ldots\|\boldsymbol{W}_{k+1}\|_*\cdot\|\boldsymbol{W}_{k-1}\|_*\ldots\|\boldsymbol{W}_1\|_*\cdot\|\boldsymbol{x}\|_2\\
&=\frac{\prod_{l=1}^L\|\boldsymbol{W}_l\|_*}{\|\boldsymbol{W}_k\|}\times\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{1}{d_L}\|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})-\boldsymbol{y}\|_2\cdot\|\boldsymbol{x}\|_2\\
&\le\frac{\prod_{l=1}^L\|\boldsymbol{W}_l\|_*}{\|\boldsymbol{W}_k\|_*}\cdot\frac{1}{\sqrt{d_L}}\sqrt{\frac{2}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\frac{1}{2d_L}\|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{w})-\boldsymbol{y}\|_2^2}\cdot\sqrt{\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\|\boldsymbol{x}\|_2^2}\\
&=\frac{\prod_{l=1}^L\|\boldsymbol{W}_l\|_*}{\|\boldsymbol{W}_k\|_*}\cdot\sqrt{\frac{2\mathcal{L}(\boldsymbol{w})}{d_L}}\cdot\sqrt{\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\|\boldsymbol{x}\|_2^2}.
\end{aligned}
$$

In the above argument, the first inequality follows by recursive application of the operator norm upper bound, and the second inequality follows from the Cauchy-Schwarz inequality. The right-hand side simplifies under Prescription 1, and we may apply Lemma 6 to obtain:

$$
\begin{aligned}
\|\nabla_{\boldsymbol{W}_k}\mathcal{L}(\boldsymbol{w})\|_F&\le\frac{\prod_{l=1}^L\|\boldsymbol{W}_l\|_*}{\|\boldsymbol{W}_k\|_*}\cdot\sqrt{\frac{2\mathcal{L}(\boldsymbol{w})}{d_L}}\cdot\sqrt{\frac{1}{|\mathsf{S}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathsf{S}}\|\boldsymbol{x}\|_2^2}\\
&\le\frac{\sqrt{d_L/d_0}}{\sqrt{d_k/d_{k-1}}}\cdot\sqrt{\frac{2}{d_L}}\cdot\sqrt{d_0}=\sqrt{2}\cdot\sqrt{\frac{d_{k-1}}{d_k}}.
\end{aligned}
$$

□

Lemma 8 (Convergence rate to critical point)**.** *Consider a fully-connected network trained by automatic gradient descent (Theorem 5) and square loss for T iterations. Let G_t denote the gradient summary (Definition 24) at step $t \le T$. Under Assumptions 1 and 2 and supposing that Prescription 1 is maintained throughout training, AGD converges at the following rate:*

$$\min_{t\in\{1,\ldots,T\}} G_t^2 \le \frac{11}{T}.$$

Proof. Theorem 5 prescribes that $\exp\eta = \frac{1}{2}(1+\sqrt{1+4G})$, and so $\eta = \log\left(1+\frac{\sqrt{1+4G}-1}{2}\right)$. We begin by proving some useful auxiliary bounds. By Lemma 7 and

Prescription 1, the gradient summary is bounded by:

$$G := \frac{1}{L}\sum_{k=1}^{L}\sqrt{d_k/d_{k-1}}\cdot\|\nabla_{\boldsymbol{W}_k}\mathcal{L}(\boldsymbol{w})\|_F \leq \frac{1}{L}\sum_{k=1}^{L}\sqrt{2} < 2.$$

The fact that the gradient summary G is less than two is important because, for $x \leq 1$, we have that $\log(1+x) \geq x\log 2$. In turn, this implies that since $G < 2$, we have that $\eta = \log\frac{1+\sqrt{1+4G}}{2} \geq \frac{\sqrt{1+4G}-1}{2}\log 2$. It will also be important to know that for $G < 2$, we have that $\frac{1}{2}\cdot G \leq \frac{\sqrt{1+4G}-1}{2} \leq G$.

With these bounds in hand, the analysis becomes fairly standard. By an intermediate step in the proof of Theorem 5, the change in objective across a single step is bounded by:

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) - \mathcal{L}(\boldsymbol{w}) &\leq -\eta\cdot G + \tfrac{1}{2}(\exp\eta - 1)^2 \\ &\leq -\tfrac{\sqrt{1+4G}-1}{2}(G\log 2 - \tfrac{1}{2}\tfrac{\sqrt{1+4G}-1}{2}) \\ &\leq -\tfrac{1}{2}\cdot(\log 2 - \tfrac{1}{2})\cdot G^2 \leq -G^2/11,\end{aligned}$$

where the second and third inequalities follow by our auxiliary bounds. Letting G_t denote the gradient summary at step t, averaging this bound over time steps and applying the telescoping property yields:

$$\min_{t\in[1,\ldots,T]} G_t^2 \leq \frac{1}{T}\sum_{t=1}^{T} G_t^2 \leq \frac{11}{T}\sum_{t=1}^{T}\mathcal{L}(\boldsymbol{w}_t) - \mathcal{L}(\boldsymbol{w}_{t+1}) = \frac{11}{T}\cdot(\mathcal{L}(\boldsymbol{w}_1) - \mathcal{L}(\boldsymbol{w}_T)) \leq \frac{11}{T},$$

where the final inequality follows by Lemma 6 and the fact that $\mathcal{L}(\boldsymbol{w}_T) \geq 0$.

□

Theorem 6 (Convergence rate to global minima)**.** *For automatic gradient descent (Theorem 5) in the same setting as Lemma 8 but with the addition of Assumption 3, the mean squared error objective at step T obeys:*

$$\mathcal{L}(\boldsymbol{w}_T) \leq \frac{1}{\alpha^2}\times\frac{6}{T}.$$

Proof. By Assumption 3, the gradient summary at time step t must satisfy $G_t \geq \alpha\times\sqrt{2\cdot\mathcal{L}(\boldsymbol{w}_t)}$. Therefore the objective at time step t is bounded by $\mathcal{L}(\boldsymbol{w}_t) \leq G_t^2/(2\alpha^2)$. Combining with Lemma 8 then yields that:

$$\mathcal{L}(\boldsymbol{w}_T) = \min_{t\in[1,\ldots,T]}\mathcal{L}(\boldsymbol{w}_t) \leq \frac{1}{2\alpha^2}\min_{t\in[1,\ldots,T]} G_t^2 \leq \frac{6}{\alpha^2 T}.$$

The proof is complete. □

H.2 PyTorch Implementation

The following code implements automatic gradient descent in PyTorch [271]. We include a single gain hyperparameter which controls the update size and may be increased from its default value of 1.0 to slightly accelerate training. We emphasise that all the results reported in the paper used a gain of unity.

```
import math
import torch

from torch.nn.init import orthogonal_

def singular_value(p):
    sv = math.sqrt(p.shape[0] / p.shape[1])
    if p.dim() == 4:
        sv /= math.sqrt(p.shape[2] * p.shape[3])
    return sv

class AGD:
    @torch.no_grad()
    def __init__(self, net, gain=1.0):

        self.net = net
        self.depth = len(list(net.parameters()))
        self.gain = gain

        for p in net.parameters():
            if p.dim() == 1: raise Exception("Biases are not supported.")
            if p.dim() == 2: orthogonal_(p)
            if p.dim() == 4:
                for kx in range(p.shape[2]):
                    for ky in range(p.shape[3]):
                        orthogonal_(p[:,:,kx,ky])
            p *= singular_value(p)

    @torch.no_grad()
    def step(self):

        G = 0
        for p in self.net.parameters():
            G += singular_value(p) * p.grad.norm(dim=(0,1)).sum()
        G /= self.depth

        log = math.log(0.5 * (1 + math.sqrt(1 + 4*G)))

        for p in self.net.parameters():
            factor = singular_value(p) / p.grad.norm(dim=(0,1), keepdim=True)
            p -= self.gain * log / self.depth * factor * p.grad
```

H.3 Other optimisers

Algorithm	Reference	# hyperparams
SGD+Armijo	Vaswani et al. [272]	5
Parabolic Line Search	Mutschler and Zell [273]	1
L4	Rolinek and Martius [274]	5
HyperSGD	Chandra et al. [275]	many
D-Adaptation	Defazio and Mishchenko [276]	1
DoG	Ivgi et al. [277]	2+
Stochastic Polyak step-size	Loizou et al. [278]	2
COCOB	Orabona and Tommasi [279]	1
Prodigy	Mishchenko and Defazio [280]	2+

Table H.1: Other approaches to optimisation without learning rates. This table lists some optimisers and the number of hyperparameters each one uses (not including optional ones like learning rate decay, or other training hyperparameters like batch size).

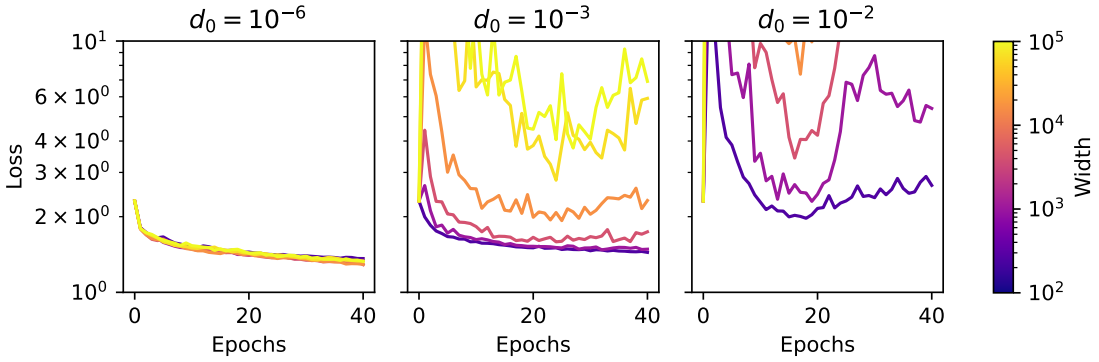


Figure H.1: The required initial step size d_0 in Prodigy is coupled to width. We trained depth-2 fully connected networks with different widths on CIFAR-10 with the Prodigy optimizer [280]. We varied the d_0 hyperparameter and for each setting of d_0 we trained networks of varying width. The plots demonstrate that the optimal setting of d_0 in Prodigy is coupled to width in an undesirable way.