Spectral Machine Learning

Diego Granziol

Christ Church

University of Oxford

A thesis presented for the degree of

Doctor of Philosophy

Michaelmas 2020

I dedicate this thesis to my mother, Manuela, whose tremendous support, love and interest in my education got me here today. I would also like to thank my father, Markus, for inspiring me to be a better version of myself. Last but not least I would like to expressly thank my Supervisor, Steve, who has backed me to the hilt against heavy head winds and whose support will not be forgotten.

Information is the resolution of uncertainty

— Claude Shannon

Acknowledgements

I would like to express my greatest thanks to my collaborators, partners and professors over the past 4 years.

Binxin Ru, who taught me the meaning of collaboration, hard-work and has been a constant source of thought provoking discussions and a major part of my personal journey and growth, I hope we continue to work together.

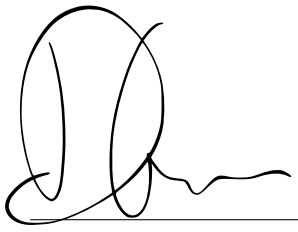
Timur Garipov, who welcomed me with open arms straight from the steak-house onto the blackboard and whose supreme coding skills in action changed my perception of what I thought was possible.

Last but certainly not least I would like to thank Xingchen Wan, my rockstar student, whose transition from undergraduate to postgraduate student has been a marvel to watch and be a part of. I want to thank him for constantly challenging my mindset and ideas, pushing me to produce my best work. Your ability to rapidly combine different ideas and keep a relevant overview at all time is absolutely incredible. I am immensely proud of our submissions and I will follow your progress with great interest.

I would also like to thank Stefan Zohren, for his exposition into random matrix theory and iterative methods, Xiaowen Dong for his graph theoretic focus, Michael Osborne for his interesting Bayesian discussions and Dmitry Vetrov, for his loss surface discussions. I'd like to also thank the members of the lab, Kyriakos, Logan, Ziahao, for creating a great atmosphere and the legacy members, Rory Beard, for all the times he helped me smash through problems, the daily discussions on Entropy and Tom Gunter, who took time out of busy schedules to show me the light and once again, Stephen for all the ideas, support, discussions and creating the atmosphere in the first place.

Declaration

I, Diego Granziol, hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.



30/12/2020

Signature *Date*

Abstract

In this thesis we develop a spectral approach to large kernel matrices, graphs and the Hessians of neural networks. This approach paves the way for efficient and improved inference schemes, novel algorithms and theoretical results. The first contribution of the thesis is the development of a novel Maximum Entropy algorithm applied to the problems of log determinant estimation (necessitated by Bayesian model selection), cluster counting and graph similarity. The method produces strong empirical results compared to existing established methods, such as the Lanczos algorithm. The second contribution is the development of a field-dependent, spiked random matrix theory for characterising the sub-sampling effects, due to mini-batching, on the loss surface of deep neural networks. The theory we develop predicts that, up to a threshold, non-adaptive-gradient methods should scale linearly with batch size and adaptive methods should scale with the square-root of the batch size. We verify these results empirically. As final contributions we develop an open source software package for neural network curvature computations, show that spectral sharpness is not relevant for solution generalisation and discuss extensions to previous theoretical work. Based upon these theoretical contributions, we develop an adaptive algorithm (Gadam) which posts strong test set performance across a variety of image and text classification problems, regularly beating finely tuned non-adaptive methods. ***Keywords***— Maximum Entropy - Stochastic trace estimation - Log Determinants - Graph clustering - Graph similarity - Neural networks - Random matrix theory - learning rate schedules - Hessian spectrum

Table of Contents

List of Figures

List of Tables

List of Abbreviations

DNN ..... Deep Neural Network

GGN ..... Generalised Gauss-Newton

STE ..... Stochastic Trace estimation

SGD ..... Stochastic Gradient Descent

RMT ..... Random Matrix Theory

Gadam ..... Adam + decoupled weight decay + iterate averaging

GadamX ..... partially adaptive Adam + decupled weight decay + iterate averaging

IA ..... Iterate Averaging

BN ..... Batch Normalisation

WD ..... Weight Decay

DOS ..... Density of States (physics talk for spectral density)

KL ..... Kullback-Leibler (divergence), also known as the relative entropy

LDL ..... Large Dimension Limit

1 | Introduction

1.1 Motivation

In the physical sciences the spectrum has been used to classify components and analyse the properties of systems with great success. In the fields of Optics, Astronomy, Condensed Matter and Quantum Physics, the spectrum is a key component in calculations and experimental verification of posited theorems. No example is more prominent than the Schrödinger equations ability to predict the spectral lines of the Hydrogen atom. To quote Richard Feynman Feynman et al. [1965]

"The most dramatic success in the history of the quantum mechanics was the understanding of the details of the spectra of some simple atoms and the understanding of the periodicities which are found in the table of chemical elements."

In this thesis we bring the spectral focus from Physics to Machine Learning. We only consider very large systems, for which the exact spectrum is inaccessible but moment-matched approximations can be inferred efficiently through the use of matrix vector multiplications. Our hypothesis is that the *properties of a system can be learned through the spectrum* and that this can be used for more efficient inference/training. We consider 3 major use cases, but believe that spectral methods constitute a canonical tool for the data sciences.

For **Bayesian inference schemes** one core computational bottleneck is the eigen-decomposition of a large kernel matrix. For **large sparse graphs**, such as social networks an eigen-decomposition of the adjacency matrix (or

Laplacian) is also infeasible. Hence, natural spectral approaches to counting the numbers of clusters and measuring graph similarity are rarely adopted. For **state of the art deep learning models**, the second derivative of the loss function is also typically only computed for very small and unrealistic, models. It is therefore difficult to validate theoretical predictions on networks and datasets of interest. From a practical standpoint it also means that more per iteration efficient optimisation schemes, which use the second derivative of the loss, are not adopted.

1.2 Thesis Outline

Chapter 2 — Introduces stochastic trace estimation, which is used as a method for garnering moment information for the spectrum of large matrices and subsequently employed in the rest of the thesis. It further introduces the field of Entropic inference, its theoretical underpinnings for estimating smooth spectral approximations, along with links to Physics and Bayesian inference.

Chapter 3 – Applies the method of maximum entropy to both graph similarity and clustering.

Chapter 4 — Introduces a software contribution to allow the spectral computation of large and expressive modern deep neural networks, along with some example code and applications. We also discuss the Lanczos algorithm in depth, along with many misconceptions in the literature.

Chapter 5 — We develop a random matrix theory framework to analyse the effect of mini-batching on curvature. We find applications in the

learning rate scaling (with batch size) of adaptive and non adaptive-gradient methods and the damping coefficients for adaptive/second-order methods.

Chapter 6 — We use our random matrix theory framework to consider the generalisation gap between adaptive and non adaptive-gradient methods

Chapter 7 — Based on the insight of the irrelevance of flatness, we propose a new optimiser Gadam and variant GadamX, which seeks sharp solutions with superior generalisation to that of Stochastic Gradient Descent. We discuss optimisation, generalisation, Polyak averaging and regularisation.

2 | Entropic Inference

In this Chapter we demonstrate the use of **Entropic inference** by considering both Bayesian and frequentist viewpoints of probability theory. We show that the **Method of Maximum Entropy** can be seen as a natural method of generating priors, which are based on constraints of a physical system. We then consider the **Method of Relative Maximum Entropy**, which we show subsumes both Bayesian inference and the method of maximum entropy.

The ideas within this Chapter led to four publications at peer-reviewed conferences [Fitzsimons et al., 2017, Granziol and Roberts, 2017, Ru et al., 2017, Granziol et al., 2019a]. In Fitzsimons et al. [2017] I supplied the theoretical backbone for the motivations of the Maximum entropy method, the concentration result depending on matrix size and the synthetic experiment. Jack Fitzsimons suggested the combination of stochastic trace estimation and Maximum Entropy and took charge of the experiment comparing the various methods, whilst Kurt Cutajar performed the GMRF experiment and Stephen Roberts was crucial in checking the validity of the theory. For Granziol and Roberts [2017] I developed the theory and ran the experiments. For Ru et al. [2017] my contribution was limited to understanding the Gaussian approximation in a general maximum entropy framework. In Granziol et al. [2019a] I developed the novel Maximum Entropy algorithm, devised the theory as well as the algorithm for calculating the moments of a Gaussian mixture. Binxin Ru took charge of the the Bayesian Optimisation experiments and provided important checks on the various other experiments and theory.

2.1 Probability Theory

In his most important piece of work "Ars Conjectandi" [Bernoulli, 1713], James Bernoulli laid down the mathematical foundation of probabilities and considered the implications of a consistent process of inference [Hon, 2008]. One of his first ideas was that if an observer had no information about the possible outcomes, then they should assign the same probability to each outcome. This was later coined the Principle of Insufficient Reason [Eva, 2019]. Generalising this idea, for a hypothesis space $H = H(A_1...A_k)$, labelling the number of cases where A is favourable (as in what we are interested in) as m and the total number of cases as n, we have as the definition of the probability of getting A or A being true

$$p(A) \equiv \frac{m}{n} \tag{2.1}$$

Given the impossibility of knowing the entire hypothesis space, which Bernoulli acknowledged by stating "what mortal will ever determine, for example, the number of diseases.." [Bernoulli, 1713] It would clearly be mad to want to learn anything this way", he made the leap between the ratio within the hypothesis space and the number of observed outcomes, ν

$$\nu(A) \equiv \frac{m}{n} \tag{2.2}$$

To determine the relationship between Equation 2.1 and Equation 2.2 one can consider the Binomial distribution

$$P(m|p,n) = \binom{n}{m} p^m (1-p)^{n-m} \tag{2.3}$$

for which the sum over $m = 0$ to $m = n$ is 1. Considering the expected number of successes

$$\langle m \rangle = \sum_{m=0}^{n} \binom{n}{m} m p^m (1-p)^{n-m} = \sum_{m=0}^{n} \frac{n!}{(n-m!)(m-1)!} p^m (1-p)^{n-m} \quad (2.4)$$

noting that the $m = 0$ term is zero, hence writing the sum from $m = 1$ and making the substitution of $n' = n - 1$ and $m' = m - 1$ we have

$$\langle m \rangle = np \sum_{m'=0}^{n'} \binom{n'}{m'} m' p^{m'} (1-p)^{n'-m'} = np \quad (2.5)$$

A similar calculation gives $\langle m(m-1) \rangle = n(n-1)p^2$ and hence combining the two we have that the expected value of the frequency is equal to the probability

$$\left\langle \frac{m}{n} \right\rangle = \frac{\langle m \rangle}{n} = p \quad (2.6)$$

and that the variance is

$$Var\left(\frac{m}{n}\right) = \frac{(m)}{n^2} = \frac{p(1-p)}{n} \quad (2.7)$$

Hence, as proposed by Bernoulli in the limit of infinite trials, the frequency equals the probability. He called this his "Golden Theorem". Mathematical extensions of this concept, where the probability is defined as the long run relative frequency, is known as the *frequentist view of probability* [Cramer, 1946, Samuels and Feller, 1973].

Problems with Frequentism: For many problems of interest, we do not have enough data or "trials" to expect the variance between the frequency and probability to be insignificant. In many reasonable questions to which we would hope the theory of probability would be able to give an answer, the

concept of an infinite set of trials is meaningless. An example of this would be deducing the probability of a particular politician winning an election. To answer this problem, do we need a (near) infinite number of (imaginary) trials with identical conditions to make a prediction? Furthermore, it can be shown that confidence intervals in the frequentist school of thought can be completely misleading and fail to include relevant prior information which can be incredibly informative in the case of limited sample data [Rosenkrantz, 1989a]. We hence consider an alternative view of probability, called the *Bayesian viewpoint*.

2.1.1 Cox's Axioms and Bayesianism

In this section, we consider whether there is a consistent way to assign probabilities based on our degrees of belief. We invoke two axioms.

Axiom 2.1. The degree of belief that a proposition a given c is true $[a|c]$, must be related to the degree of belief that *not* a given c $[\hat{a}|c]$ is true

$$P(\tilde{a}|c) = f(P(a|c)) \tag{2.8}$$

where $f(.)$ is a monotonic function, this requirement stems from transitivity. If the probability of a, $P(a)$ is greater than $P(b)$ and $P(b) > P(c)$, then $P(a) > P(c)$. If this is not true, then we cannot rank and thus cannot compare the degrees of belief.

Axiom 2.2. The degree of belief, that a and b, given c are true $[a, b|c]$, must be related to $[a|c]$ and $[b|ac]$ We consider this axiom reasonable. If a is true, then we must investigate whether b is true, given a and c. The discussion as to why this does not depend on all the possible prepositions is discussed elsewhere [Tribus, 1969, Smith and Erickson, 1990, Caticha, 2012].

By representing degrees of belief as real numbers, and re-graduating the scale, Cox [1946] was able to attain the identities satisfied by all theories of probability not violating his axioms,

$$P(a, b|c) = P(a|c)P(b|a, c) \tag{2.9}$$

$$P(a|c) + P(\tilde{a}|c) = 1 \tag{2.10}$$

Where Equation 2.9 is known as the product rule and Equation 2.10 is known as the sum rule. By the symmetry of the second axiom, i.e. the interchangeability of b and c mean that

$$P(b|a, c) = \frac{P(b|c)P(a|b, c)}{P(a|c)} \tag{2.11}$$

A result obtained, but not explicitly justified by Laplace, known as *Bayes theorem* [Joyce, 2004]. Removing the conditionality on c we have the somewhat more familiar form.

$$P(b|a) = \frac{P(b)P(a|b)}{P(a)} \tag{2.12}$$

This is known as marginalisation. From the sum rule, i.e. Equation 2.10 we note that $\sum_a P(a|c) = 1$ hence applying this to the sum over b to the product rule Equation 2.9,

$$\sum_b P(a, b|c) = \sum_b P(a|c)P(b|a, c) = P(a|c) \sum_b P(b|a, c) = P(a|c) \tag{2.13}$$

where we have taken the first term out of the sum as it does not depend on b and simply applied the sum rule, where we consider the joint event $a, c = c'$ to equal some new event without loss of generality. We can hence rewrite

Bayes theorem as:

$$P(b|a) = \frac{P(b)P(a|b)}{\sum_b P(a|b)P(b)} \tag{2.14}$$

where $P(b)$ denotes the prior, $P(a|b)$ the likelihood, $P(b|a)$ the posterior and the normalising factor in the denominator the evidence.

2.1.1.1 Discussion on Subjective Probabilities

The notion of a subjective probability and some of the frequentist criticisms thereof were somewhat alleviated by the work of De Finetti [1974] by showing that the probability axioms of Kolmogorov could be derived through manipulating probabilities in such a manner to ensure a gambling system based on them would not result in a sure loss.

This is known as the *Dutch book* argument. Such a system of probability satisfied non negativity, normalisation and finite additivity. A special case of this very general framework of probability is the Bayesian probability [Walley, 1991]. Such notions have allowed for a demonstration that the axioms of Quantum Mechanics are nothing more than the Bayesian theory of probability generalised to the complex space of Hermitian matrices $\mathcal{C}_h^{n\times n}$ [Benavoli et al., 2016].

What about priors? Neither the logicist, nor the subjectivist, gives a metric of optimality when choosing priors. The goal is to make the best possible inference given the information at hand. We thus seek a consistent and principled way to select priors. Furthermore, we seek a justification for the principle of indifference, which assigns a flat prior to the set of possibilities in a state of maximum ignorance. In our next section, we introduce the concept of Entropy, which forms the basis for the the *method of maximum entropy*.

As a further point, our arguments and axiomatic justifications have been based on events. What if we have information not in the form of events or trials? What if we know that the mean of a random variable is a certain fixed value? How do we incorporate this into our inference? We show that Bayes rule alone is unable to deal with such constraint information and show that a more general method, the *method of maximum relative entropy*, which subsumes both maximum entropy and Bayes rule as special cases, can be used.

2.2 Shannon-Boltzmann-Gibbs Entropy

Following on from our previous discussion, the greatest degree of belief that we can assign to an event A_1 over the event space is the probability $p(A) = 1$. This means we are completely certain that event A_1 will take place. We can extend this notion by assigning a very high probability to A_1 and a very low probability to the other A_i's. Whilst we are not certain that event A_1 is going to happen, we can conclude that we are very confident about this. How confident we are depends on the size of the event space and how evenly we distribute the probabilities across the event space. To lay the mathematical foundations of this idea, that more peaked distributions have less uncertainty, Shannon derived a unique information measure $\mathcal{S}$ satisfying the following axioms.

Axiom 2.3. $\mathcal{S}$ is a real continuous function of the probabilities.

Axiom 2.4. If all the p_i's are equal then $\mathcal{S} = F(n)$ which is a monotonically increasing function of n

Axiom 2.5. For all possible groupings $g = 1...N$ of the states $i = 1...n$ we must have

$$\mathcal{S}[P] = \mathcal{S}_g[P] + \sum_g P_g \mathcal{S}_g[p|g] \tag{2.15}$$

Assuming all states are equally likely so $p_i = 1/n$. There are N groups g with the same number of states $m = n/N$, noting that the sum over P_g gives 1.

$$F(mn) = F(n) + F(m) \tag{2.16}$$

For any two integers $t, s > 1$, the ratio of their logairithms can be approximated arbitrarily closely by a rational number.

$$\frac{\alpha}{\beta} \leq \frac{\log s}{\log t} \leq \frac{\alpha+1}{\beta} \equiv t^\alpha \leq s^\beta \leq t^{\alpha+1} \tag{2.17}$$

Requiring that F is monotonically increasing and using Equation 2.16, we can approximate $F(s)$ and $F(t)$ by the same rational number $\alpha \div \beta$,

$$F(t^\alpha) \leq F(s^\beta) \leq F(t^{\alpha+1}) \Rightarrow \alpha F(t) \leq \beta F(s) \leq (\alpha+1)F(t) \Rightarrow \left| \frac{F(s)}{F(t)} - \frac{\log s}{\log t} \right| \leq \frac{1}{\beta} \tag{2.18}$$

For sufficiently large β we have $F(s)/\log s = k$. Substituting this into Axiom 2.5, we obtain:

$$\begin{aligned} \mathcal{S}_G[P] &= F(n) - \sum_g P_g F(m_g) = \sum_g P_g [F(n) - F(m_g)] \\ &= \sum_g P_g k \log \frac{n}{m_g} = -k \sum_{i=1}^{N} P_g \log P_g \end{aligned} \tag{2.19}$$

where we assume the groups g have different sizes m_g, with $P_g = m_g/n$. This information measure $\mathcal{S}$ is thus identical to that of the thermodynamic entropy and hence the concatenation of the name the Boltzmann-Shannon-

Gibbs (BSG) entropy.

2.2.1 Combinatorial Basis of Entropy

Consider N particles in s distinguishable sub-groups, where each particle within each sub-group is indistinguishable. The number of ways a given such a configuration can be realised, i.e. different *microstates* for the same *macrostate* is

$$W(N_k) = \frac{N!}{n_1!n_2!.....n_s!} \tag{2.20}$$

where we have n_1 particles in group 1 etc. The logarithm of Equation 2.20 can be written using Stirling's approximation [Mermin, 1984] to second order

$$\log W(N_k) \approx -N\left(\sum_i \frac{n_i}{N}\log(\frac{n_i}{N})\right) + \frac{1}{2}\log\left(\frac{N}{(2\pi)^{s-1}\Pi_i n_i}\right) \tag{2.21}$$

We note that the Shannon entropy is the first term in Equation 2.21. It can be shown using induction that Equation 2.20 ≥ 1 and similarly that the second term in Equation 2.21 ≤ 0. The second term, which we drop in subsequent analysis thus acts to lower the number of ways a set of configurations can be realised. We note that in the large N limit, the number of ways a configuration can be realised can be accurately approximated as $W(N_k) \approx \exp(N\mathcal{S})$. It can be seen that if we consider the configuration that maximizes $\mathcal{S}$ and consider the fraction of all possible configurations that it occupies, in the large N limit we find that

$$\frac{W(N_{\mathcal{S}})}{\sum_k W(N_k)} = \frac{\exp(N\mathcal{S}_{max})}{\sum_k \exp(N\mathcal{S}_k)} \xrightarrow[N\to\infty]{limit} 1 \tag{2.22}$$

This large N limit is known as the Thermodynamic limit [Chandler, 1987]. This argument is originally due to Boltzmann [Landau et al., 2012] and in-

volves considering the phase space of a single particle, divided into s cells, each with energy ϵ_i and where i denotes the particular microstate. Boltzmann assumed that each cell had an equal chance of being occupied by a given particle and hence the distribution for a particular macrostate was proportional to its multiplicity, which is given by Equation 2.20. He then maximised the multiplicity (or the log thereof which is equivalent due to the monotonicity of the log), given his knowledge of the system. A closed system has a fixed total number of particles N and energy E, therefore the constraints can be written as $\sum_i^s n_i = N$ and $\sum_i^s n_i \epsilon_i = E$. Using the generic method of Lagrange multipliers, one can solve this equation to derive the canonical distribution. Extensions by Gibbs allow for the constraint of a mean energy and particle number, as opposed to fixed, leading to the grand canonical ensemble.

2.3 Method of Maximum Entropy

By considering Shannon's information measure as the unique measure of uncertainty about a distribution, Jaynes argued that the least biased probability assignment that could be made was by finding the probabilities that maximised the functional given the known constraints [Jaynes, 1957b]. By considering constraints as a form of information and considering the idea that all information reduces uncertainty, choosing any other distribution would amount to making an arbitrary extra assumption. This gives rise to the idea that maximising the entropy is equivalent to maximising the uncertainty. Hence, given information in the form of mean value constraints and a particular domain of applicability, we maximise the entropic functional $\mathcal{S}$, which is a function of the probability density $p(x)$

$$\mathcal{S} = -\sum_{x\in\mathbb{D}} p(x)\log p(x)dx - \alpha\left[\sum_{x\in\mathbb{D}} p(x) - 1\right] - \sum_i \lambda_i\left[\sum_{x\in\mathbb{D}} p(x)f_i(x)dx - \mu_i\right]. \tag{2.23}$$

In which μ_i are the mean-value constraints and λ_i are the Lagrange multipliers to be determined via optimisation of the entropic functional, it can be seen that for the constraint of an expected value of energy $f_i(x) = \epsilon(x)$ one recovers the canonical ensemble and that for an additional expected particle number value one recovers the grand canonical ensemble. Beyond showing that the entirety of statistical mechanics could be considered as a particular case of a general method of inference, irrespective of ergodicity, metric transitivity and the assumption of equal a priori probabilities, Jaynes demonstrated (by analysis of Wolf's dice data), that erroneous predictions when using **MaxEnt** meant that important constraint information had been left out and could be used to deduce new physics [Rosenkrantz, 1989b]. Jaynes was further able to show the extent of possible distributions allowed by the constraint within an entropy $\Delta\mathcal{S}$ of the maximum, scaled as $\Delta\mathcal{S} = (2N)^{-1}\chi^2_d(1-F)$, where N denotes the number of trials (or die tosses), F is the significance, so 0.95 corresponds to 95% and χ^2_d is the chi-squared with d degrees of freedom [Jaynes, 1983].

Why Mean Value Constraints? From a Physics perspective, mean value constraints follow from the grand canonical ensemble [Landau et al., 2012], where particles can interchange but their mean number is conserved. Indeed one of Jaynes great achievements was to consider statistical mechanics from an information theoretic perspective [Jaynes, 1957a]. For our purposes, it is prudent to consider whether more general inequality constraints [Boyd and

Vandenberghe, 2009] could be more appropriate. Mathematically this would correspond to using the KKT conditions over Lagrange multipliers. Our reasons for not investigating this extension in depth were largely practical. Our applications were largely focused on stochastic trace estimation of matrix powers. Whilst bounds for stochastic trace estimations exist [Hutchinson, 1990b], the bounds are incredibly loose and in practice we found the variance and hence uncertainty which would feed into a practical bound to be very small and hence did not seem a natural extension of our work. Indeed in section 2.5, we actually prove that for large matrices, which are of interest to us, that we expect the standard deviation in the estimates to become small in relation to the expectation, justifying their neglection in the anaylsis.

2.4 Maximum Entropy (MaxEnt) Algorithm

In this section, we develop an algorithm for determining the Maximum Relative Entropy density given moment constraints. Under the assumption of a flat prior in a bounded domain, this reduces to a generic MaxEnt density, which we use in various examples. We relegate a discussion on the Maximum Relative entropy and its derivation to Appendix A, as it does not form a core part of our learning procedure. The main change is that if we wish to incorporate prior information, we must alter the entropic functional to the form

$$S[p(x), q_0(x)] = -\int p(x) \log p(x) dx + \int p(x) \log q_0(x) dx \tag{2.24}$$

where $q_0(x)$ is our prior, taken to be flat for MaxEnt. As we discussed previously, in order to obtain the MaxEnt distribution, we maximise the generic objective of Equation 2.24, under constaints as in Equation 2.23.

In practice, we instead minimise the dual form of this objective [Boyd and Vandenberghe, 2009], which can be written as follows:

$$\mathcal{S}(q, q_0) = \int_{x \in \mathcal{D}} q_0(x) \exp(-[1 + \sum_i \alpha_i x^i]) dx + \sum_i \alpha_i \mu_i. \tag{2.25}$$

with a very similar equation appearing in Caticha [2012] Where the α_i correspond to the λ_i in Equation 2.23 and denote the constraints. Notice that this dual objective admits analytic expressions for both the gradient and the Hessian, with respect to the undetermined parameters:

$$\frac{\partial \mathcal{S}(q, q_0)}{\partial \alpha_j} = \mu_j - \int_{x \in \mathcal{D}} q_0(x) x^j \exp(-[1 + \sum_i \alpha_i x^i]) dx, \tag{2.26}$$

$$\frac{\partial^2 \mathcal{S}(q, q_0)}{\partial \alpha_j \partial \alpha_k} = \int_{x \in \mathcal{D}} q_0(x) x^{j+k} \exp(-[1 + \sum_i \alpha_i x^i]) dx. \tag{2.27}$$

For a flat prior in a bounded domain, which we use as a prior for the spectral densities of large matrices and for Gaussian mixture models, $q_0(x)$ is a constant that can be dropped out.

A flat prior in for the eigenvalue density of a matrix is at first glance unconvincing, as Weyl's law [Weyl, 1912] characterises the decay of the eigenvalues (which is far from uniform) and there are known analytic forms for the spectra of many classes random matrices, such as the semi-circle and Marchenko-Pastur laws [Akemann et al., 2011] (which are also far from uniform on the support). However we note that unlike in typical Bayesian analysis, where the practitioners sets a prior, with a given strength (think of a Gaussian with a very small variance which in the limit becomes a delta function and hence no data influences the prior), in MaxEnt the distribution must conform to the constraints. In fact the constraints determine the distirbution to such a

large extent that in our practical experiments we found no noteable difference and more significantly, no practical improvement between using a flat or non-flat prior with support over the domain.

With the notation

$$q_\alpha(x) = \exp(-[1 + \sum_{i=0}^{m} \alpha_i x^i]), \tag{2.28}$$

we obtain the MaxEnt algorithm detailed in Algorithm 1.

Algorithm 1 The Proposed MaxEnt Algorithm

1: **Input:** Moments $\{\mu_i\}$, Tolerance Level ϵ,Hesian Jitter $\eta = 10^{-8}$
2: **Output:** Coefficients $\{\alpha_i\}$
3: Initialize $\alpha_i = 0$
4: Compute gradient: $g_j = \mu_j - \int_0^1 q_\alpha(x) x^j dx$
5: Compute Hessian: $H_{jk} = \int_0^1 q_\alpha(x) x^{j+k} dx$, $H = \frac{1}{2}(H + H^T) + \eta I$
6: Minimize $\int_0^1 q_\alpha(x) dx + \sum_i \alpha_i \mu_i$ using Conjugate Gradient until $\forall j$: $g_j < \epsilon$

The input moments $\{\mu_i\}$ of Algorithm 1 are estimated using fast moment estimation techniques, as explained in Section 2.5. In our implementation, we use Python's SciPy Newton-conjugate gradient (CG) algorithm to solve the minimisation in step 6, having firstly computed the gradient within a tolerance level of ϵ as well as the Hessian. To make the Hessian better conditioned so as to achieve convergence, we symmetrise it and add jitter[1] of intensity $\eta = 10^{-8}$ along the diagonal. We estimate the given integrals with quadrature using a grid size of 10^{-4}. Empirically, we observe that the algorithm is not overly sensitive to these choices. In particular, we find that, for well conditioned problems, the need for jitter is obviated and a larger grid interval works fine; in the case of worse conditioning, jitter helps improve convergence and the grid size becomes more important, where a smaller grid size leads to a computationally more intensive implementation.

[1] $\boldsymbol{H} \rightarrow \boldsymbol{H} + \eta \boldsymbol{I}$

Given that any polynomial sum can be written as $\sum_i \alpha_i x^i = \sum_i \beta_i f_i(x)$, where $f_i(x)$ denotes another polynomial basis and $\alpha_i, \beta_i \in \mathbb{R}$, whether we choose to use power moments in our constraints or another polynomial basis, such as the Chebyshev or Legendre basis, does not change the entropy or solution of our objective. However, the performance of optimisers working on these different formulations may vary [Skilling, 1989]. For simplicity, we have kept all the formulas in terms of power moments. However, we find vastly improved performance and conditioning when we switch to orthogonal polynomial bases (so that the errors between moment estimations are uncorrelated). We implement both Chebyshev and Legendre moments in our Lagrangian and find similar performance for both. The theory of orthogonal polynomials is well developed and understood [Szeg, 1939] and similar remarks have been made in Skilling [1989].

Although not known at the time of writing, we note that other Maximum Entropy algorithms were made available around a similar time Phillips et al. [2017]. An identical software setup to ours was later also republished in Saad and Ruai [2019], indicating that despite the apparently simple setup and solution of the algorithm, that either the combination had not been tried before or was not easy for researchers to find. An alternative hypothesis is that there was a limited distribution of up to date MaxEnt software.

2.5 Stochastic Trace Estimation

Stochastic trace estimation is an effective way of cheaply estimating the moments of the spectral density of a given matrix $\boldsymbol{H} \in \mathbb{R}^{n \times n}$ [Hutchinson, 1990b]. The essential idea is that we can estimate the non-central moments of the spectrum by using matrix-vector multiplications. Recall that for any

multivariate random variable $\mathbf{z}$ with mean $\mu_{\mathbf{z}}$ and variance Σ, we have:

$$\mathbb{E}(\mathbf{z}^T\mathbf{z}) = {\mu_{\mathbf{z}}}^T\mu_{\mathbf{z}} + \Sigma \xrightarrow[\mu_{\mathbf{z}}=\mathbf{0}]{\Sigma=I} I, \tag{2.29}$$

where for the second equality we have assumed that $\mathbf{z}$ possesses zero mean and unit variance. By the linearity of trace and expectation, for any number of moments $m \geq 0$ we have:

$$\sum_{s=1}^{n} \lambda_s^m = \mathrm{Tr}(\boldsymbol{I}\boldsymbol{H}^m) = \mathbb{E}(\mathbf{z}\boldsymbol{H}^m\mathbf{z}^T), \tag{2.30}$$

where $\{\lambda_s\}$ are the eigenvalues of K. In practice, we approximate the expectation in Equation 2.30 with a set of probing vectors and a simple Monte Carlo average, i.e., for d random unit vectors $\{\mathbf{z_j}\}_{j=1}^{d}$:

$$\mathbb{E}(\mathbf{z}\boldsymbol{H}^m\mathbf{z}^T) \approx \frac{1}{d}\left(\sum_{j=1}^{d} \mathbf{z}_j\boldsymbol{H}^m\mathbf{z}_j^T\right), \tag{2.31}$$

where we take the product of the matrix $\boldsymbol{H}$ with the iteratively updated vector $\mathbf{z}_j^T$ for m times, so as to avoid the costly matrix multiplication with $\mathcal{O}(n^3)$ complexity. This allows us to calculate the non-central moment expectations in $\mathcal{O}(dmn^2)$ complexity for dense matrices, or $\mathcal{O}(dm \times n_{nz})$ complexity for sparse matrices, where $d \times m \ll n$ and n_{nz} is the number of non-zero entries in the matrix. The random vector $\mathbf{z}_j$ can be drawn from any distribution, such as a Gaussian. However, the Hutchinston estimator, for which each element of the vector $\boldsymbol{z}_j$, is drawn from the distribution where each element is ± 1 with equal probability, can be shown to be the estimator of minimum variance which has zero bias [Hutchinson, 1990a]. In practice we do not see significant difference practically between the various choices of random vectors. The resulting stochastic trace estimation algorithm is outlined in

Algorithm 2.

Algorithm 2 Stochastic Trace Estimation

Input: Matrix $\boldsymbol{H}^{n\times n}$, Moments m, Number of Probe Vectors d
Output: Moment Expectations $\mu_i = \mathbb{E}_p(\lambda^i)$, $\forall i \leq m$
Initialise $\mathbb{E}_p(\lambda^i) = 0\ \forall i$
for $j = 1, \ldots, d$ **do**
  Initialise $\mathbf{z}_j = \text{rand}(1, n)$
  for $i = 1, \ldots, m$ **do**
    $\mathbf{z}_j^T = \boldsymbol{H}\mathbf{z}_j^T$
    $\mathbb{E}_p(\lambda^i) = \mathbb{E}_p(\lambda^i) + \frac{1}{d}\mathbf{z}_j\mathbf{z}_j^T$
  end for
end for

How many random vectors do we need? In Algorithm 2 a key hyper-parameter is the number of probe vectors used, d. The time complexity of the algorithm is proportional to this number. However, too few vectors and the moment estimates may be too noisy. Hence we present a result that we believe is novel. For large matrices we expect the noise to signal ratio to reduce approximately inversely to the matrix dimension.

Lemma 1. *Let $\boldsymbol{u} \in \mathbb{R}^{P\times 1}$ be a random vector, where $\boldsymbol{u}_i$ has zero mean and unit variance and finite 4'th moment $\mathbb{E}[\boldsymbol{u}_i^4] = m_4$. Then for all symmetric $\boldsymbol{H} \in \mathbb{R}^{P\times P}$*

- $\mathbb{E}[\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{u}] = \text{Tr}\,\boldsymbol{H}$
- $\text{Var}[\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{u}] \leq (2 + m_4)\,\text{Tr}(\boldsymbol{H}^T\boldsymbol{H})$

Proof.

$$\mathbb{E}[\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{u}] = \sum_{i,j=1}^{P} \boldsymbol{H}_{i,j}\mathbb{E}[\boldsymbol{u}_i\boldsymbol{v}_j] = \sum_{i=1}^{P} \boldsymbol{H}_{i,i} = \text{Tr}\,\boldsymbol{H} \tag{2.32}$$

$$\begin{aligned}\mathbb{E}[||\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{u}||^2] &= \sum_{i,j}\sum_{k,l}\boldsymbol{H}_{i,j}\boldsymbol{H}^T_{k,l}\mathbb{E}[\boldsymbol{u}_i\boldsymbol{u}^T_j\boldsymbol{u}_k\boldsymbol{u}^T_l] \\ &= \sum_{i,j}\sum_{k,l}\boldsymbol{H}_{i,j}\boldsymbol{H}^T_{k,l}[\delta_{i,j}\delta_{k,l}+\delta_{i,l}\delta_{j,k}+\delta_{i,k}\delta_{j,l}+m_4\delta_{i,j,k,l}] \quad (2.33) \\ &= (\mathrm{Tr}\,\boldsymbol{H})^2+(2+m_4)\,\mathrm{Tr}(\boldsymbol{H}^2)\end{aligned}$$

□

Remark. *Let us consider the signal to noise ratio for* $\boldsymbol{H} \succ 0$*, where* $\succ$ *denotes positive definiteness of the matrix*

$$\left(\frac{\sqrt{\mathrm{Var}[\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{u}]}}{\mathbb{E}[\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{u}]}\right)^2 \propto \frac{1}{1+\frac{\sum_{i\neq j}^{P}\lambda_i\lambda_j}{\sum_k^P\lambda_k^2}} = \frac{1}{1+\frac{P-1\langle\lambda_i\lambda_j\rangle}{\langle\lambda_k^2\rangle}} \leq \frac{1}{1+\frac{P-1}{\kappa^2}} \quad (2.34)$$

where $\langle..\rangle$ denotes the arithmetic average λ_i the i'th eigenvalue, P the matrix dimension and $\kappa = \lambda_1/\lambda_P$ is the condition number. The derivation follows directly from 1 and neglecting the constant factor $2+m_4$ which is bounded, along with noting that $\mathrm{Tr}(\boldsymbol{H}^2) = \mathrm{Tr}(U\boldsymbol{H}^2U) = \sum_i^P\lambda_i^2$. For the extreme case of all eigenvalues being identical, this reduces to $1/P \to 0$ in the $P\to\infty$ limit, whereas for a rank-1 matrix, this ratio remains 1. For the Marchenko-Pastur density, which is regularly used as a typical density representing arbitary covariance matrices [Bun et al., 2017] and detailed in depth in Chapter 4, κ is not a function of P as $P\to\infty$ and hence in many situation of interest (For example estimating the spectra of neural networks, which we do in Chapter 4, the Marchenko-Pastur density has been used to model the spectrum theoretically [Pennington and Worah, 2018]) we also expect this benign dimensional scaling to apply. The calculation that the variance for d random vectors is reduced by a factor of d is omitted.

2.6 Maximum Entropy and Log Determinants

Any symmetric positive-definite (PD) matrix K is diagonalisable by a unitary matrix U, i.e., $K = U^T DU$, where D is the diagonal matrix of eigenvalues of K. Hence we can write the log determinant as:

$$\log\det(K) = \log\prod_i \lambda_i = \sum_{i=1}^{n} \log\lambda_i = n\mathbb{E}_p(\log\lambda), \tag{2.35}$$

where we have used the cyclicity of the determinant and $\mathbb{E}_p(\log\lambda)$ denotes the expectation under the spectral measure , which is given by $p(\lambda)$ .The latter can be written as:

$$\mathbb{E}_p(\log\lambda) = \int_{\lambda_{\min}}^{\lambda_{\max}} p(\lambda)\log\lambda d\lambda = \int_{\lambda_{\min}}^{\lambda_{\max}} \sum_{i=1}^{n} \frac{1}{n}\delta(\lambda-\lambda_i)\log\lambda d\lambda, \tag{2.36}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ correspond to the largest and smallest eigenvalues, respectively.

Given that the matrix is PD, we know that $\lambda_{\min} > 0$ and we can divide the matrix by an upper bound of the eigenvalue, $\lambda_{\max} \leq \lambda_u$, via the Gershgorin circle theorem [Gershgorin, 1931] such that:

$$\log\det(K) = n\mathbb{E}_p(\log\lambda') + n\log\lambda_u, \tag{2.37}$$

where $\lambda' = \lambda/\lambda_u$ and $\lambda_u = \max_i(\sum_{j=1}^{n}|K_{ij}|)$, i.e., the maximum sum of the rows of the absolute of the matrix K. Hence we can comfortably work with the transformed measure:

$$\int_{\lambda_{\min}/\lambda_u}^{\lambda_{\max}/\lambda_u} p(\lambda')\log\lambda' d\lambda' = \int_0^1 p(\lambda')\log\lambda' d\lambda', \tag{2.38}$$

where the spectral density $p(\lambda')$ is 0 outside of its bounds of $[0, 1]$. We therefore arrive at the following Maximum Entropy Method (MEMe) for log determinant estimation, detailed in Algorithm 3.

Algorithm 3 MEMe for Log Determinant Estimation

1: **Input:** PD Symmetric Matrix $K^{n\times n}$, Number of Moments m, Number of Probe Vectors d, Tolerance Level ϵ
2: **Output:** Log Determinant Approximation $\log\det(K)$
3: $\lambda_u = \max_i(\sum_{j=1}^{n} |K_{ij}|)$
4: $B = K/\lambda_u$
5: μ (moments)$\leftarrow$ StochasticTraceEstimation(B, m, d) via Algorithm 2
6: α (coefficients) $\leftarrow$ MaxEnt(μ, ϵ) via Algorithm 1
7: $q(\lambda) \leftarrow q_\alpha(\lambda) = \exp\left[-(1 + \sum_i \alpha_i \lambda^i)\right]$
8: $\log\det(K) \leftarrow n \int \log(\lambda) q(\lambda) d\lambda + n \log(\lambda_u)$

2.6.1 Experiments

In the following where possible we use absolute relative estimation error which is defined as

$$\frac{|V_{true} - V_{est}|}{V_{true}} \tag{2.39}$$

Where V_{true}, V_{est} are the true and estimated values of a given quantity (in the next subsection log determinants). We use this value because we are not interested in the sign of the error and we want an error as a fraction of the true value.

Log Determinant Estimation for Synthetic Kernel Matrices: In order to specifically compare against commonly used Chebyshev [Han et al., 2015] and Lanczos approximations [Ubaru et al., 2016] to the log determinant, and see how their accuracy degrades with worsening condition number, we generate a typical squared exponential kernel matrix [Rasmussen, 2006],

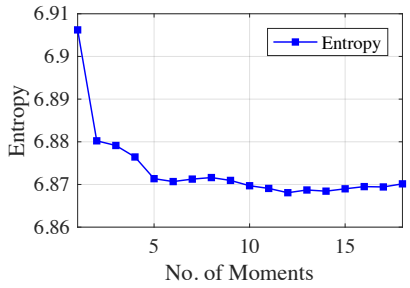
$K(\boldsymbol{x}_1, \boldsymbol{x}_2) \in \mathbb{R}^{1000\times1000}$ (we take the number to simulate a large matrix but to allow easy full eigendecomposition to calculate the ground truth), using the Python GPy package with 6 dimensional Gaussian inputs ($\boldsymbol{x}$) with a variety of realistic uniform length-scales. We then add noise of variance $\eta = 10^{-8}$ along the diagonal of K. We use $m = 30$ moments and $d = 50$ Hutchinson probe vectors [Hutchinson, 1990b] in MEMe for the log determinant estimation. We display the absolute relative estimation error for different approaches in Table 2.1. We see that, for low condition numbers, the benefit of framing the log determinant as an optimisation problem is marginal, whereas for large condition numbers, the benefit becomes substantial, with orders of magnitude better results than competing methods.

Table 2.1: Relative estimation error for MEMe, Chebyshev, and Lanczos approaches, with various length-scale l and condition number κ on the squared exponential kernel matrix $K \in \mathbb{R}^{1000\times1000}$.

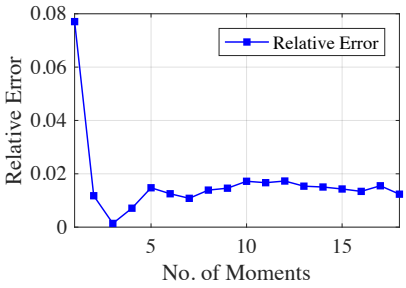
κ	l	MEMe	Chebyshev	Lanczos
3×10^1	0.05	**0.0014**	0.0037	0.0024
1.1×10^3	0.15	0.0522	**0.0104**	0.0024
1.0×10^5	0.25	0.0387	0.0795	**0.0072**
2.4×10^6	0.35	0.0263	0.2302	**0.0196**
8.3×10^7	0.45	**0.0284**	0.3439	0.0502
4.2×10^8	0.55	**0.0256**	0.4089	0.0646
4.3×10^9	0.65	**0.00048**	0.5049	0.0838
1.4×10^{10}	0.75	**0.0086**	0.5049	0.1050
4.2×10^{10}	0.85	**0.0177**	0.5358	0.1199

We note that the exponential covariance function, which has known analytical forms for its eigenvalues and eigenfunctions [Zhu et al., 1997a], for which the eigenvalues of the kernel matrix give an approximation thereof [Rasmussen, 2006], that the eigenvalues have an exponential decay and hence we expect an exponential spectral density to fit well. There is of course an inherent smoothness assumption in the Method of Maximum Entropy, where

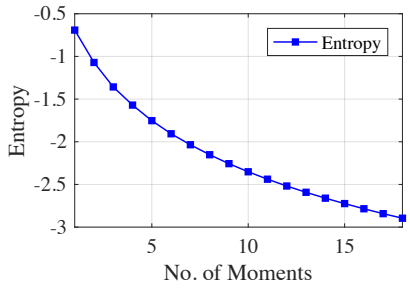
by smoothness we mean the minimum KL divergence between the density satisfying the constraints and the uniform.



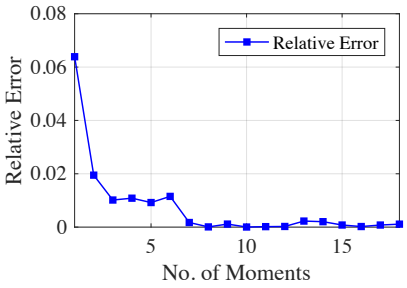
(a) BMxnt: Entropy vs Moments



(b) BMxnt: Relative Error vs Moments



(c) MEMe: Entropy vs Moments



(d) MEMe: Relative Error vs Moments

Figure 2.1: Comparison of the classical (BMxnt) and proposed (MEMe) MaxEnt algorithms for log determinant estimation on real data.

Log Determinant Estimation on Real Data: The addition of an extra moment constraint cannot increase the entropy of the MaxEnt solution Cover and Thomas [2012], Zhu et al. [1997b]. For the problem of log determinants, this signifies that the entropy of the spectral approximation should decrease with the addition of every moment constraint. We implement the MaxEnt algorithm proposed in Bandyopadhyay et al. [2005], which we refer to as BMxnt (an abbreviation for Bandyopadhyay-MaxEnt), in the same manner as applied for log determinant estimation in Fitzsimons et al. [2017], and compare it against the proposed MEMe approach. Specifically, we show

results on data from the Ecology dataset [Leskovec and Krevl, 2014], with $n = 80900$, for which the true log determinant can be calculated. For BMxnt, we see that after the initial decrease, the error (Figure 2.1b) begins to increase for $m > 4$ moments and the entropy (Figure 2.1a) starts to increase from $m > 12$ moments. Given that the maximum entropy solution can only have a decreased entropy for more constraints, this indicates an optimisation problem. For the proposed MEMe algorithm, the performance is vastly improved in terms of estimation error (Figure 2.1d); furthermore, the error continually decreases with increasing number of moments, and the entropy (Figure 2.1c) decreases smoothly. This demonstrates the superiority both in terms of consistency and performance of our algorithm compared to established alternatives. We note from the work of Caticha [2012], that the differential entropy (required when the density considered is continuous) depends on the discretization interval and hence that actual value of entropy are not meaningful, but only differences in entropy. We could have alternatively instead of using the value given by the inner optimisation loop, which in unaffeted by a constant shift, normalised both for the 2 moment case to the analytical differential entropy of the Gaussian. This was not done, firstly because it is not clear that the two algorithms would have both given this exact value and secondly we were only interested in observing a monotonic decrease in entropy for increasing moment number.

2.6.2 Spectral Density and Eigenvalue Distribution

Given that a specific matrix has a spectral density of delta functions at the locations $\lambda_1 \geq \lambda_2 \geq ...$ it may seen strange to talk about a spectral density and relate that to the distribution of maximum entropy, given that we have not defined the random variable defined with this distribution. Note

the similarity to quantum mechanics, where the energies of particles in the system are the eigenvalues of the Hamiltonian (which is a Hermitian operator). Whilst for a given system each particle has a specific energy λ_i, we can consider the system as a draw from an imaginary ensemble of systems with the same mean energy and hence the spectral density becomes a distribution over the real line. For our case the draw is a kernel or other associated matrix drawn from the data generating process. It can be shown for many classes of random matrices, that the spectra of random matrices concentrate around their expectation in the limit of large dimension [Bun et al., 2017]. We might consider how to relate the joint probability density function of all the eigenvalues $p(\lambda_1, \lambda_2, ..\lambda_P)$ to the spectral density (or simply counting function of the eigenvalues in a certain interval) $n(x) = 1/P \sum_i^P \delta(\lambda - \lambda_i)$.

$$\langle n(x) \rangle := \int .. \int d\vec{\lambda} p(\lambda_1, .., \lambda_P) = \frac{1}{P} \sum_i^P \int .. \int d\vec{\lambda} p(\lambda_1, ..., \lambda_P.) \delta(\lambda - \lambda_i) = p(\lambda) \tag{2.40}$$

and hence the average spectral density (over the ensemble) is equal to the marginal density $p(\lambda)$. So on our deeper level our justification for MaxEnt is that we have decided to swap the unknown counting function for a specific matrix, with its mean under the data generating process. In order to answer how the moments of a specific matrix are likely to represent those of the ensemble average, for matrices with independent entries a_{ij}, $\forall i > j$, with common element wise bound K, common expectation μ, variance σ^2 and diagonal expectation $\mathbb{E}(a_{ii}) = \nu$, it can be shown that the corrections to the semi-circle law for the moments of the eigenvalue distribution

$$\langle \mu_n, \lambda^m \rangle = \int_{\mathcal{R}} \lambda^m d\mu(x) = \frac{1}{P} \mathrm{Tr} X_n^m \tag{2.41}$$

have a corrective factor bounded by Füredi and Komlós [1981]

$$\frac{K^2 m^6}{2\sigma^2 P^2}. \tag{2.42}$$

Hence, the finite size effects are larger for higher moments than the lower counterparts, further justifying the use of using a limited number of moments. Similarly, considering the spectral moments of a given random matrix and their expectation under the generating process, $\langle \hat{\mu_n}, x^m \rangle$. It can be easily seen using Chebyshev's inequality that

$$\mathbb{P}\big(|\langle \mu_n, x^m \rangle - \langle \hat{\mu}_n, x^m \rangle| > \epsilon\big) \leq \frac{1}{\epsilon^2} \Big| \mathbf{E}\big(\langle \mu_n, x^m \rangle\big)^2 - \big(\mathbf{E}\langle \mu_n, x^m \rangle\big)^2 \Big| \tag{2.43}$$

writing this in terms of matrix traces we have

$$\frac{1}{P^2}\Big[\mathbf{E}((\mathrm{Tr}X_P^m)^2 - (\mathbf{E}\mathrm{Tr}X_P^m)^2\Big] = \frac{1}{P^2}\sum_{i,i'}\big[\mathbf{E}\zeta_i\zeta_{i'} - \mathbf{E}\zeta_i\mathbf{E}\zeta_{i'}\big] \tag{2.44}$$

where ζ_i is shorthand for the product $\zeta_{i_1 i_2}...\zeta_{i_m i_1}$ with $i_1, ..., i_m \in \{1, ..., n\}$, where each pair i, i' generates a graph with vertices $V_{i,i'} = \{i_1, ..i_m\} \cup \{i'_1, ..., i'm\}$ and edges $E_{i,i'} = \{i_1 i_2, .., i_m i_1\} \cup \{i'_1 i'_2, ..., i'_m i'_1$. We consider the entries of the Matrix X to be zero mean and independent, with bounded moments. For the term in equation 2.44 corresponding to i, i' to be non-zero, each edge must appear at least twice, since the entries of X_n have zero mean and the graphs must have at least one edge in common otherwise equation 2.44 is 0 by independence. The *weight* of (i, i') is defined as the cardinality of $V_{i,i'}$, with pairs (i, i') and (j, j') said to be equivalent if there is a bijection mapping coresponding indices to each other. It can be easily shown that there are no non-zero pairs of weights $t \geq m + 1$. By the boundedness of the moments of the entries of M_P the contribution of each term is

$\mathcal{O}(1/P^{m+2})$ so the greatest corrective term for large P is $t = m$ for which there are equivalent pairs [Feier, 2012]

$$w = P(P-1)..(P-t+1) = \frac{P!}{(P-t)!} \tag{2.45}$$

using Stirlings approximation so the total correction goes as

$$\sum_{t=1}^{m} \frac{1}{P^{m-t+2}} \exp\left(P \sum_{i=2}^{\infty} \left(\frac{t}{n}\right)^{i} \frac{1}{i(i+1)}\right) \tag{2.46}$$

again we note the corrective terms increase with the number of moments m. Hence, for finite P, which subsumes the set of all real networks, the lower graph spectral moments are more representative to those of the underlying stochastic process and closer to those of the limiting spectral density than their higher counterparts.

What does the assumption of a flat prior on the spectral density correspond to? For a group of particles, the assumption of equal energy under no extra information seems reasonable. However for matrices i.i.d Gaussian elements give the semi-circle law, an outer product of Gaussian matrices give the Marchenko-Pastur law. One could consider a random matrix where all elements are equal and set to the value of a bounded random variable, which is chosen from the uniform distribution. The average spectral density would be uniform (single eigenvalue at a given number, all others zero). As discussed previously the effects of the constraints are so overwhelming that the choice of any prior with support on the domain is in all practical cases irrelevant. The main advantage which we exploit is the existence of a unique smooth spectral density approximation, which could be satisfied for any prior $q(x)$, which for simplicity we choose constant.

3 | Graph similarity and Clustering

In this Chapter we apply the principle of Maximum Entropy, discussed in Chapter 2 to the problem of spectral graph estimation. Whilst the approximate spectrum may be of general interest, we specifically focus on the applications to determining the number of clusters in a graph and graph similarity. Much of this work appeared in the PrePrint of Granziol et al. [2018]. In this work I devised the theory, paper structure the application domain of graph similarity and BinXin Ru helped with the implementation of the baselines, experiments and checked the theory and text. Xiaowen Dong identitied the usefulness of MaxEnt for cluster number counting and Stefan Zohren introduced the links between random matrix theory and graphs.

3.1 Relevance of Graphs and their spectra

Many systems of interest can be naturally characterised by complex networks; examples include social networks [Mislove et al., 2007, Flake et al., 2000, Leskovec et al., 2007], biological networks [Palla et al., 2005] and technological networks. Trends, opinions and ideologies spread on a social network, in which people are nodes and edges represent relationships. Networks are mathematically represented by graphs. Of crucial importance to the understanding of the properties of a network or graph is its spectrum, which is defined as the eigenvalues of its adjacency or Laplacian matrix [Farkas et al., 2001, Cohen-Steiner et al., 2018]. The spectrum of a graph is the density of the eigenvalues of its adjacency/Laplacian matrix. The spectrum of a graph can be considered as a natural set of graph invariants and has been extensively studied in the fields of chemistry, physics and mathematics [Biggs

et al., 1976]. Spectral techniques have been extensively used to characterise the global network structure [Newman, 2006b] and in practical applications thereof, such as facial recognition and computer vision, learning dynamical thresholds, clustering [Von Luxburg, 2007], and measuring graph similarity [Takahashi et al., 2012].

A major limitation in utilising graph spectra to solve problems such as graph similarity and estimating the number of clusters[1] is the inability to automatically and consistently learn an everywhere-positive, non-singular approximation to the spectral density. Full eigendecomposition, which is prohibitive for large graphs, or iterative moment-matched approximations both give a Dirac sum that must be smoothed to be everywhere positive. The choice of smoothing kernel $k_\sigma(x, x')$ and kernel bandwidth choice σ, or number of histogram bins, which are usually chosen in an ad-hoc manner, can significantly affect the resulting output.

The main contributions of this Chapter are as follows:

- We prove that the method of kernel smoothing, commonly used in methods to visualise and compare graph spectral densities, biases moment information;
- We propose a computationally efficient and information theoretically optimal smooth spectral density approximation, based on the method of Maximum Entropy, which fully respects the moment information. It further admits analytic forms for symmetric and non-symmetric KL-divergences and Shannon entropy;
- We utilise our information theoretic spectral density approximation, on

[1] Just two example applications of the general method we propose for learning graph spectrum in this paper.

two example applications. We investigate graph similarity and to learn the number of clusters in a graph, outperforming iterative smoothed spectral approaches on both real and synthetic data-sets

3.2 Preliminaries

Graphs are the mathematical structure underpinning the formulation of networks. Let $G = (V, E)$ be an undirected graph with vertex set $V = \{v_i\}_{i=1}^{n}$. Each edge between two vertices v_i and v_j carries a non-negative weight $w_{ij} > 0$. $w_{ij} = 0$ corresponds to two disconnected nodes. For un-weighted graphs we set $w_{ij} = 1$ for two connected nodes. The *adjacency matrix* is defined as $\mathbf{W}$ and $w_{ij} = [\mathbf{W}]_{ij}$. The degree of a vertex $v_i \in V$ is defined as

$$d_i = \sum_{j=1}^{n} w_{ij}. \tag{3.1}$$

The *degree matrix* $\mathbf{D}$ is defined as a diagonal matrix that contains the degrees of the vertices along diagonal, i.e., $\mathbf{D}_{ii} = d_i$. The *unnormalised graph Laplacian* matrix is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{W}. \tag{3.2}$$

As G is undirected, $w_{ij} = w_{ji}$, which means that the weight matrix is symmetric and hence $\mathbf{W}$ is symmetric and given $\mathbf{D}$ is symmetric, the unnormalised Laplacian is also symmetric. As symmetric matrices are special cases of normal matrices, they are Hermitian matrices and have real eigenvalues. Another common characterisation of the Laplacian matrix is the *normalised*

Laplacian [Chung, 1997],

$$\mathbf{L}_{\text{norm}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{W}_{\text{norm}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}, \tag{3.3}$$

where $\mathbf{W}_{\text{norm}}$ is known as the normalised adjacency matrix[2]. The spectrum of the graph is defined as the density of the eigenvalues of the given adjacency, Laplacian or normalised Laplacian matrices corresponding to the graph. Unless otherwise specified, we will consider the spectrum of the normalised Laplacian.

3.3 Motivation

For large sparse graphs with millions, or billions, of nodes, learning the exact spectrum using eigen-decomposition is unfeasible due to the $\mathcal{O}(n^3)$ cost. Powerful iterative methods (such as the Lanczos algorithm, Kernel polynomial methods (KPM), Chebyshev/Taylor approximations, the Haydock method and many more) only require matrix-vector multiplications, and hence have a computational cost scaling with the number of non-zero entries in the graph matrices, are often used. There is extensive literature documenting the performance of these methods. Lin et al. [2016] state that the Lanczos algorithm is significantly more accurate than other methods (including the KPM), followed by the Haydock method. Ubaru et al. [2017] show that the convergence of the Lanczos algorithm is twice that of the Chebyshev approximation. Hence given the superior theoretical guarantees and empirical performance of the Lanczos algorithm, we employ it as a sole baseline against our MaxEnt method. The Lanczos algorithm approximates the graph spectrum with a sum of weighted Dirac delta functions, closely matching the first

[2]Strictly speaking, the second equality only holds for graphs without isolated vertices.

m moments (where m is the number of iterative steps) of the spectral density Ubaru et al. [2016]:

$$p(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\delta(\lambda - \lambda_i) \approx \sum_{i=1}^{m} w_i \delta(\lambda - \lambda_i), \tag{3.4}$$

where $\sum_{i=1}^{m} w_i = 1$, and λ_i denotes the i-th eigenvalue in the spectrum. However, such an approximation is undesirable because natural divergence measures between densities, such as the information-based relative entropy $\mathcal{D}_{\mathrm{KL}}(p||q) \in (0, \infty)$ [Cover and Thomas, 2012], as Equation 3.5,

$$\mathcal{D}_{\mathrm{KL}}(p||q) = \int p(\lambda) \log \frac{p(\lambda)}{q(\lambda)} d\lambda, \tag{3.5}$$

can be infinite for densities $p(\lambda), q(\lambda)$ that are mutually singular. The use of the Jensen-Shannon divergence simply re-scales the divergence into $\mathcal{D}_{\mathrm{JS}}(p||q) \in (0, 1)$. This can lead to counter-intuitive scenarios, such as an infinite (or maximal) divergence upon the removal or addition of a single edge or node in a large network, an infinite (or maximal) divergence between two graphs generated using the same random graph model and identical hyper-parameters. From our previous section, we can see that although the empirical spectral density is a sum of delta functions, the average spectral density over the ensemble of networks generated using the same stochastic process, likely will not be. Take for example a generating process which generates two graphs at random $\mathcal{G}_1, \mathcal{G}_2$ in which every node is paired with another node with probability p (an Erdős-Rényi graph), we would like some measure which indicates that both graphs are very similar, they both come from the same generating process. The KL divergence, which is a distance between distributions satisfying many uniqueness properties [Cover and Thomas, 2012], makes sense when considering the average spectral densities (which we estimate using the

moments of the empirical spectral densities), but not the empirical spectral densities themselves. As we will see in the next section, many practictioners through use of histograms or kernel smoothing, already do this implicitly in an ad-hoc manner.

3.3.1 The argument against kernel smoothing:

To alleviate these limitations, practitioners typically generate a smoothed spectral density by convolving the Dirac mixture with a smooth kernel $k_\sigma(\lambda - \lambda_i)$ [Takahashi et al., 2012], often a Gaussian or Cauchy [Banerjee and Jost, 2008] to facilitate visualisation and comparison. The smoothed spectral density $\tilde{p}(\lambda)$, with reference to Equation 3.4, thus takes the form:

$$\begin{aligned} \tilde{p}(\lambda) &= \int k_\sigma(\lambda - \lambda')p(\lambda')d\lambda' \\ &= \int k_\sigma(\lambda - \lambda') \sum_{i=1}^{n} w_i \delta(\lambda' - \lambda_i)d\lambda' = \sum_{i=1}^{n} w_i k_\sigma(\lambda - \lambda_i) \end{aligned} \tag{3.6}$$

We make some assumptions regarding the nature of the kernel function, $k_\sigma(\lambda - \lambda_i)$, in order to prove our main theoretical result about the effect of kernel smoothing on the moments of the underlying spectral density. Both of our assumptions are met by (the commonly employed) Gaussian kernel.

Assumption 1. *The kernel function $k_\sigma(\lambda - \lambda_i)$ is supported on the real line $[-\infty, \infty]$.*

Assumption 2. *The kernel function $k_\sigma(\lambda - \lambda_i)$ is symmetric and permits all moments.*

Theorem 2. *The m-th moment of a Dirac mixture $\sum_{i=1}^{n} w_i \delta(\lambda - \lambda_i)$, which is smoothed by a kernel k_σ satisfying assumptions 1 and 2, is perturbed from its unsmoothed counterpart by an amount $\sum_{i=1}^{n} w_i \sum_{j=1}^{r/2} \binom{r}{2j} \mathbb{E}_{k_\sigma(\lambda)}(\lambda^{2j}) \lambda_i^{m-2j}$,*

where $r = m$ if m is even and $m-1$ otherwise. $\mathbb{E}_{k_\sigma(\lambda)}(\lambda^{2j})$ denotes the $2j$-th central moment of the kernel function $k_\sigma(\lambda)$.

Proof. The moments of the Dirac mixture are given as,

$$\langle \lambda^m \rangle = \sum_{i=1}^{n} w_i \int \delta(\lambda - \lambda_i)\lambda^m d\lambda = \sum_{i=1}^{n} w_i \lambda_i^m. \tag{3.7}$$

The moments of the modified smooth function (Equation 3.6) are

$$\begin{aligned} \langle \tilde{\lambda}^m \rangle &= \sum_{i=1}^{n} w_i \int k_\sigma(\lambda - \lambda_i)\lambda^m d\lambda = \sum_{i=1}^{n} w_i \int k_\sigma(\lambda')(\lambda' + \lambda_i)^m d\lambda' \\ &= \langle \lambda^m \rangle + \sum_{i=1}^{n} w_i \sum_{j=1}^{r/2} \binom{r}{2j} \mathbb{E}_{k_\sigma(\lambda)}(\lambda^{2j}) \lambda_i^{m-2j}. \end{aligned} \tag{3.8}$$

Here we have used the binomial expansion and the fact that the infinite domain is invariant under shift reparametarisation and the odd moments of a symmetric distribution are 0. □

Remark. *The above proves that kernel smoothing alters moment information, and that this process becomes more pronounced for higher moments. Furthermore, given that $w_i > 0$, $\mathbb{E}_{k_\sigma(\lambda)}(\lambda^{2j}) > 0$ and for the normalised Laplacian $\lambda_i > 0$, the corrective term is manifestly positive, so the smoothed moment estimates are biased.*

Remark. *For large random graphs, the moments of a generated instance converge to those averaged over many instances, hence by biasing our moment information we limit our ability to learn about the underlying stochastic process.*

3.4 Information Theoretic Approach

For large, sparse graphs corresponding to real networks with millions or billions of nodes, where eigen-decomposition is intractable, we may still be able to compute a certain number of matrix-vector products, which we can use to get unbiased estimates of the spectral density moments, using stochastic trace estimation (as explained in Section 2.5). We can settle on a unique spectral density which satisfies the given moment information exactly, known as the density of Maximum Entropy explained in Section 2.4. The resultant entropic spectral density has the form

$$p(\lambda|\{\alpha_i\}) = \exp[-(1 + \sum_i \alpha_i \lambda^i)], \tag{3.9}$$

where the coefficients $\{\alpha_i\}_{i=1}^m$ are derived from Algorithm 1. For simplicity, we denote $p(\lambda|\{\alpha_i\}_{i=1}^m)$ as $p(\lambda)$. [3]

Algorithm 4 Entropic Graph Spectrum (EGS) Learner.

1: **Input:** Normalised Laplacian $\mathbf{L}$, number of probe vectors d, number of moments used m
2: **Output:** EGS $p(\lambda)$
3: Moments $\{\mu_i\}_{i=1}^m \leftarrow$ STE $(\mathbf{L}, d, m)$
4: MaxEnt Coefficients $\{\alpha_i\}_{i=1}^m \leftarrow$ MaxEnt Algorithm $(\{\mu_i\}_{i=1}^m)$
5: Entropic graph spectrum $p(\lambda) = \exp[-(1 + \sum_i \alpha_i \lambda^i)]$

3.4.1 The Entropic Graph Spectral Learning algorithm

The full algorithm for learning the Entropic Graph Spectrum (EGS) is summarized in Algorithm 4. We first estimate the m moments of the normalised

[3] We make our Python code available on https://github.com/diegogranziol/Python-MaxEnt.

graph Laplacian $\{\mu_i\}_{i=1}^m$ via STE, then use the moment information to solve for MaxEnt coefficients $\{\alpha_i\}_{i=1}^m$ and compute the EGS via Equation 3.9.

3.5 Visualising the Modelling Power of EGS

Having developed a theory as to why a smooth, exact moment-matched approximation of the spectral density is crucial to learning the characteristics of the underlying stochastic process and having proposed a method (Algorithm 4) to learn such a density, we test the practical utility of our algorithm on examples where the limiting spectral density is known.

3.5.1 Erdős-Rényi graphs and the semi-circle law

For Erdős-Rényi graphs with edge creation probability $p \in (0, 1)$, despite overloading the use of the symbol p we keep it here because of its consistent use in the graph literature, hence we distinguish between p in the context of random graphs, which determines the connection probability between two nodes and $p(\lambda)$ which denotes the spectral density. As long as $np \to \infty, n \to \infty$, the limiting spectral density of the normalised Laplacian converges to the semi-circle law and its Laplacian converges to the free convolution of the semi-circle law and $\mathcal{N}(0, 1)$. We consider here to what extent our EGS algorithm can effectively approximate this density with finite moments. Wigner's density is fully defined by its infinite number of central moments given by $\mathbb{E}_\mu(\lambda^{2n}) = (R/2)^{2n} C_n$, where $C_n \times (n + 1) = \binom{2n}{n}$ are known as the Catalan numbers. As a toy example we generate a semi-circle centred at $\lambda = 0.5$ with $R = 0.5$ and use the analytical moments to compute its corresponding EGS. As can be seen in Figure 3.2a, for $m = 5$ moments, the central portion of the density is already well approximated, but the end

points are not. This is largely corrected for $m = 30$ moments. To further see how the fit improves with the number of moments we plot the KL divergence between the semi circle density and the MaxEnt fit in 3.2b. We generate an Erdős-Rényi graph with $p = 0.001$ and $n = 5000$, and learn the moments using stochastic trace estimation. We then compare the fit between the EGS computed using a different numbers of input moments $m = 3, 30, 60, 100$ and the graph eigenvalue histogram computed by eigen-decomposition. We plot the results in Figure 3.1. One striking difference between this experiment and the previous one is the number of moments needed to give a good fit. This can be seen especially clearly in the top left subplot of Figure 3.1, where the 3 moment, i.e. Gaussian approximation, completely fails to capture the bounded support of the spectral density. Given that the exponential polynomial density is positive everywhere, it needs more moment information to learn the regions of boundedness of the spectral density in its domain. In the previous example we artificially alleviated this phenomenon by putting the support of the semi-circle within the entire domain. It can be clearly seen that increasing moment information successively improves the fit to the support Figure 3.1. Furthermore, the magnitude of the oscillations, which are characteristic of an exponential polynomial function, decay in magnitude for larger moments.

3.5.2 Beyond the semi-circle law

For the adjacency matrix of an Erdős-Rényi graph with $p \propto 1/n$, the limiting spectral density does not converge to the semi-circle law and has an elevated central portion and the scale free limiting density converges to a triangle like distribution. For other random graphs, such as the Barabási-Albert (also known as the scale-free network), the probability of a new node being con-

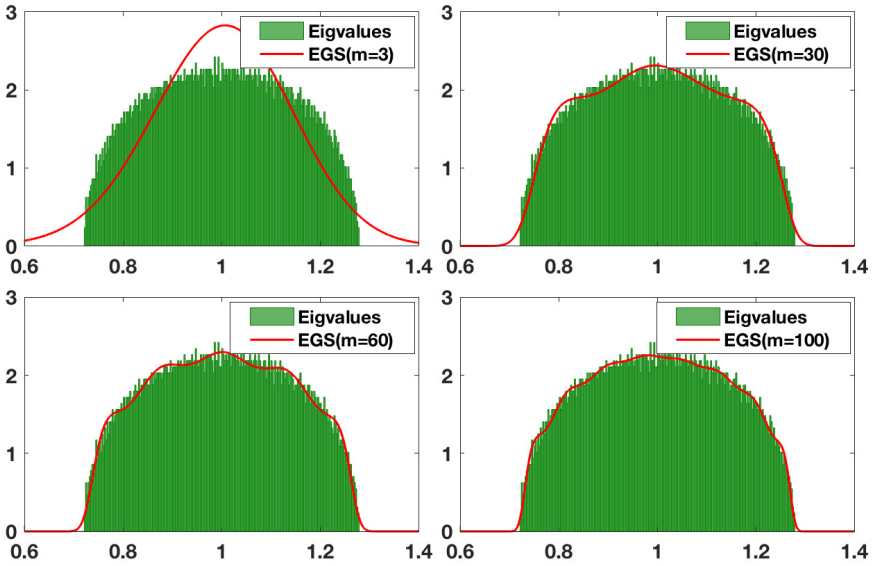
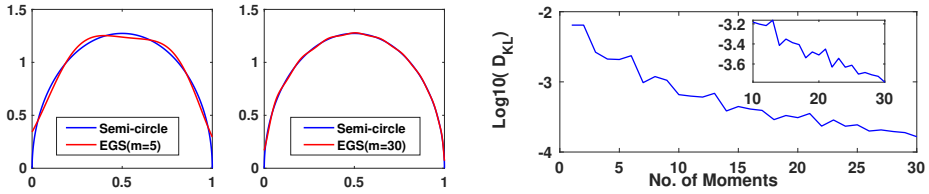


Figure 3.1: **Comparison of EGS fit to eigenvalue Histogram on ER graph:** EGS fit to randomly generated $p = 0.001, n = 5000$ Erdős-Rényi graph. The number of moments m used increases from 3 to 100 and the number of bins used for the eigenvalue histogram is $n_b = 500$. x-axis,y-axis corresponds to $\lambda, p(\lambda)$ respectively.

nected to a certain existing node is proportional to the number of links that existing node already has, violating the independence assumption required to derive the semi-circle density. We plot a Barabási-Albert network ($n = 5000$) and, similar to Section 3.5.1, we learn the EGS and plot the resulting spectral density against the eigenvalue histogram, as shown in Figure 3.3. For the Barabási-Albert network, due to the extremity of the central peak, a much larger number of moments is required to get a reasonable fit. We also note that increasing the number of moments is akin to increasing the number of bins in terms of spectral resolution, as seen in Figure 3.3.



(a) EGS fit to a semi-circle density for different moment number m.

(b) the KL divergence between the true semi-circle density and the EGS

Figure 3.2: Fit between EGS and the semi-circle distribution

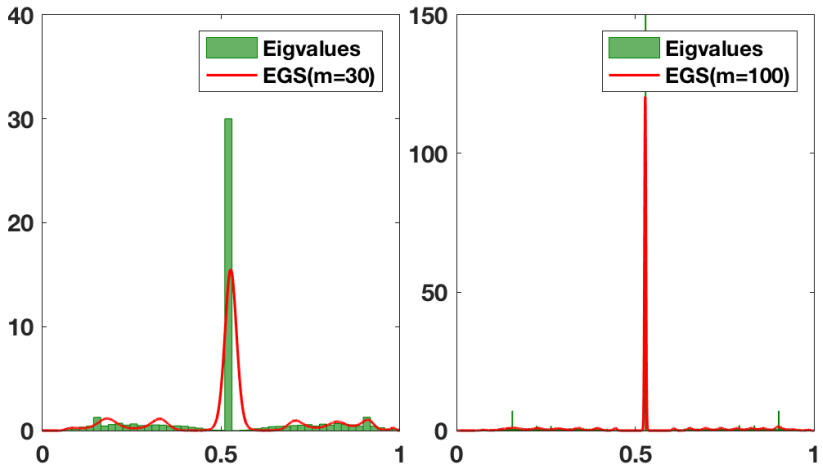


Figure 3.3: EGS fit to randomly generated $n = 5000$ Barabási-Albert network. The number of moments used for computing EGSs and the number of bins used for the eigenvalue histogram are $m = 30$, $n_b = 50$ (Left) and $m = 100$, $n_b = 500$ (Right).

3.6 EGS for Measuring Graph Similarity

In this section we test the use of our EGS, in combination with symmetric KL divergence, to measure similarity between different types of synthetic and real-world graphs. Note that our proposed EGS, based on the MaxEnt distribution, enables the symmetric KL divergence to be computed analytically.

To calculate the differential entropy we simply note that

$$\mathcal{S}(p) = \int p(\lambda) \left(1 + \sum_{i}^{m} \alpha_i \lambda^i\right) d\lambda = 1 + \sum_{i}^{m} \alpha_i \mu_i. \tag{3.10}$$

The KL divergence between two EGSs, $p(\lambda) = \exp[-(1 + \sum_i \alpha_i \lambda^i)]$ and $q(\lambda) = \exp[-(1 + \sum_i \beta_i \lambda^i)]$, can be written as,

$$\mathcal{D}_{\mathrm{KL}}(p||q) = \int p(\lambda) \log \frac{p(\lambda)}{q(\lambda)} d\lambda = -\sum_{i} (\alpha_i - \beta_i) \mu_i^p, \tag{3.11}$$

where μ_i^p refers to the i-th moment constraint of the density $p(\lambda)$. Similarly, the symmetric-KL divergence can be written as,

$$\frac{\mathcal{D}_{\mathrm{KL}}(p||q) + \mathcal{D}_{\mathrm{KL}}(q||p)}{2} = \frac{\sum_i (\alpha_i - \beta_i)(\mu_i^q - \mu_i^p)}{2}, \tag{3.12}$$

where all the α and β are derived from the optimisation and all the μ are given by stochastic trace estimation. We first investigate the feasibility of recovering the parameters of random graph models and then move onto classifying the network type as well as computing graph similarity among various synthetic and real-world graphs.

3.6.1 Inferring parameters of random graph models

We investigate whether one can recover the network parameter values of a graph via its learned EGS. We generate a random graph using the NetworkX package [Hagberg et al., 2008] of a given size and parameter value (e.g., $n = 50, p = 0.6$) and learn its associated MaxEnt spectrum using our EGS learner (Algorithm 4). Then, we generate another graph of the same size but learn its parameter value by minimising the symmetric-KL divergence

between its MaxEnt spectrum and that of the original graph. For simplicity we use a dense grid search (spacing 0.001) on the values of p. We repeat the above procedures for different random graph models i.e. Erdős-Rényi (ER), Watts-Strogatz (WS) and Barabási-Albert (BA) and different graph sizes ($n = 50, 100, 150$); results are shown in Table 3.1a. It can be seen that, given the approximate EGS, we are able to learn well the parameters of the graph producing that spectrum.

Table 3.1: Learning Random Network Parameters and Real Network Types using EGS

(a) Average parameters estimated by our MaxEnt-based method for the 3 types of network. The number of nodes in the network is denoted by n.

n	50	100	150
ER ($p = 0.6$)	0.600	0.598	0.604
WS ($p = 0.4$)	0.468	0.454	0.414
BA ($r = 0.4n$)	18.936	40.239	58.428

(b) Minimum KL divergence between the EGSs of random networks and that of a large BA graph and YouTube network.

	Large BA	YouTube
ER	2.662	7.728
WS	7.612	9.735
BA	**2.001**	**7.593**

3.6.2 Learning real-world network types

Determining which random graph model best fits a real-world network, characterised by spectral divergence, can lead to better understanding of its dynamics and characteristics. This has been explored for small biological networks [Takahashi et al., 2012] where full eigen-decomposition is viable. Here, we conduct similar experiments for large networks based on our EGS method. We first test on a large (5000-node) synthetic BA network. By minimising the symmetric KL divergence between its EGS and those of small (1000-node) random networks (ER, WS, BA), we successfully recover its own type. As a real-world use case, we further repeat the experiment to determine which

random network can best model the YouTube network and find, as shown in Table 3.1b, that the BA gives the lowest divergence.

3.6.3 Comparing different real-world networks

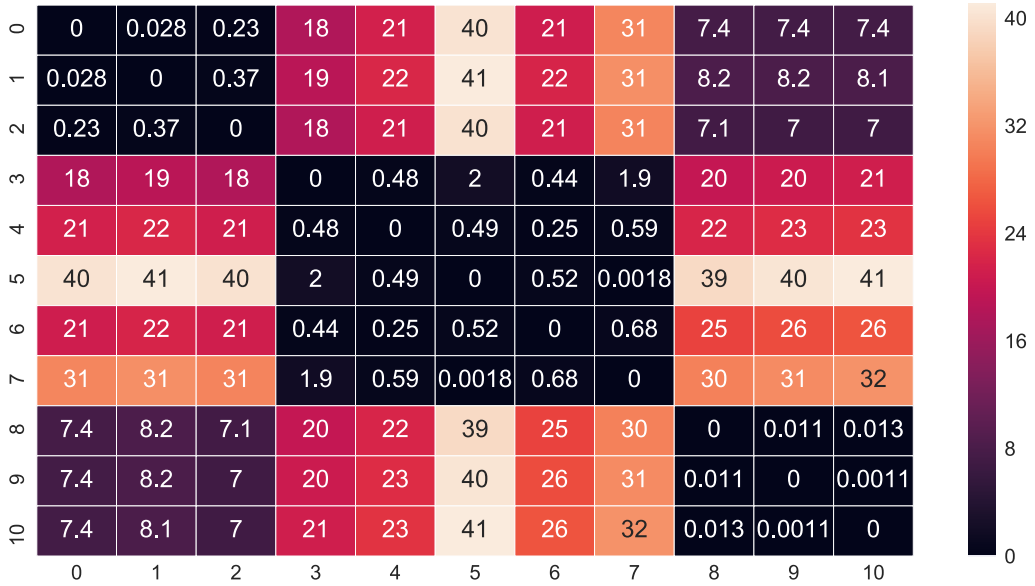


Figure 3.4: **Similar graphs have small KL divergences between their EGS:** Symmetric KL heatmap between 9 graphs from the SNAP dataset: (0) bio-human-gene1, (1) bio-human-gene2, (2) bio-mouse-gene, (3) ca-AstroPh, (4) ca-CondMat, (5) ca-GrQc, (6) ca-HepPh, (7) ca-HepTh, (8) roadNet-CA, (9) roadNet-PA, (10) roadNet-TX.

We now consider the feasibility of comparing real-world networks using EGSs. Specifically, we take 3 biological networks, 5 citation networks and 3 road networks from the SNAP dataset [Leskovec and Krevl, 2014], and compute the symmetric KL divergences between their EGS with $m = 100$ moments. We present the results in a heat map (Figure 3.4). We can see very clearly that the intra-class divergences between the biological, citation and road networks are much smaller than their inter-class divergences. This strongly suggests that the combination of our EGS method and the symmetric KL

divergence can be used to identify similarity in networks. Furthermore, as can be seen in the divergence between the human and mouse network, the spectra of human genes are more closely aligned with each other than they are with the spectra of mouse genes. This suggests a reasonable amount of intra-class distinguishability as well.

For an illuminating discussion on which graphs are determined by their spectrum, we recommend Van Dam and Haemers [2003]. Whilst it is conjectured that in the $P \to \infty$ limit that all graphs are determined by their spectrum, such a proof remains elusive. The most well known example of different graphs corresponding to identical spectra is the Saltire pair, which has $P = 5$ vertices. We furthermore consider it reasonable to expect similar generative processes to generate similar graphs. We have already shown in the previous section that for random matrices (which result from random graphs) drawn from the same generative process that we expect the differences between the lower moments to be smaller for finite P. Due to the scaling invariance $1/\sqrt{P}$ in the derivations, we expect the spectra of large graphs from the same process (even of different size) to hence have small divergences between their EGS and this is what we observe experimentally. Future work could look at developing theoretical guarantees for this method. However we note that other than very simple random graphs, which are uninteresting practically as they do not demonstrate much real world behaviour (such as hubs and the small world phenomenon), closed forms for the average spectral densities have not been discovered.

3.7 EGS for Estimating Cluster Number

It is known from spectral graph theory, that the number multiplicity of the 0 eigenvalue in the Laplacian (and the normalised Laplacian) is equal to the number of connected components in the graph. For a small amount of inter-cluster connectivity, by matrix perturbation theory we should expect a number of eigenvalues close to 0. We make this argument precise with the following Theorem 3.

Theorem 3. *The normalised Laplacian $\hat{\boldsymbol{L}}$ of a given graph $\mathcal{G}$ with corresponding eigenvalues $\lambda_1, \lambda_2, ..\lambda_P$, which is then perturbed by adding a single edge between nodes 1 and $m+1$ from two previously disconnected clusters A and B, giving a new normalised Laplacian $\hat{\boldsymbol{L}}'$ or a peturbed graph $\mathcal{G}'$, with eigenvalues $\lambda'_1, \lambda'_2, ..\lambda'_P$, has its change in eigenvalues $|\lambda_i - \lambda'_i| \, \forall \, i$ bounded to first order by*

$$\left| \frac{1}{\sqrt{d_1 d_a}} + \frac{1}{\sqrt{d_{m+1} d_b}} - \frac{2}{\sqrt{d_1 d_{m+1}}} \right|, \tag{3.13}$$

where d_i denotes the degree of node i and $1/\sqrt{d_a} = \sum_{g \in (g,1)} 1/\sqrt{d_g}$ and similarly $1/\sqrt{d_b} = \sum_{g \in (g,m+1)} 1/\sqrt{d_g}$, where $\sum_{g \in (g,1)}$ denotes the sum over all nodes connecting to node 1 and g refers to the graph.

Proof. Using Weyl's bound on Hermination matrices, where $\boldsymbol{L}_{G'} \& \boldsymbol{L}_G$ denote the perturbed and unperturbed graph (where by perturbation we mean the addition of a single node) respectively

$$\Delta\lambda_i = |\lambda'_i - \lambda_i| \leq ||\mathbf{L}_{G'} - \mathbf{L}_G||. \tag{3.14}$$

By the definition of the normalised Laplacian $\tilde{L}_{i,j} = L_{i,j}/\sqrt{d_i d_j}$

$$\Delta\tilde{L} = \sum_{g\in(g,1)} \left(\frac{1}{\sqrt{d_1 d_g}} - \frac{1}{\sqrt{(d_1+1)d_g}} \right) - \frac{2}{\sqrt{d_1 d_{m+1}}} + \sum_{g\in(g,m+1)} \left(\frac{1}{\sqrt{d_{m+1} d_g}} - \frac{1}{\sqrt{(d_{m+1}+1)d_g}} \right),$$

to first order in the binomial expansion. We hence have the result. □

Corollary 3.1. *For two clusters with identical degree $d \gg 1$, connected by a single inter-cluster link, the zero eigenvalue is perturbed to first order by at most $\Delta\lambda_0 = \frac{1}{d}$.*

Remark. *Hence for E inter-cluster connections, our bound scales as E/d and hence the intuition of a small change in the 0 eigenvalue holds if the number of edges between clusters is much smaller than the degree of the nodes within the clusters.*

3.7.1 Learning the number of clusters in large graphs

For the case of large sparse graphs, where only iterative methods such as the Lanczos algorithm can be used, the same arguments from Section 3.3 apply. This is because the Dirac delta functions are now weighted and to obtain a reliable estimate of the eigengap, one must smooth the delta functions. We would expect a smoothed spectral density plot to have a spike near 0. We expect the moments of the spectral density to encode this information and the mass of this peak to be spread. We hence look for the first spectral minimum in the EGS and calculate the number of clusters as shown in Algorithm 5. We conduct a set of experiments to evaluate the effectiveness of our spectral method in Algorithm 5 for learning the number of distinct clusters in a

network, comparing against the Lanczos algorithm with kernel smoothing on both synthetic and real-world networks.

Algorithm 5 Cluster Number Estimation.

Input: Normalised graph Laplacian $\boldsymbol{L}_{\text{norm}} \in \mathbb{R}^{n \times n}$, tolerance η
Output: Number of clusters N_c
EGS $p(\lambda) \leftarrow$ Algorithm 4()
Find minimum λ_* that satisfies $\frac{dp(\lambda)}{d\lambda}|_{\lambda=\lambda_*} \leq \eta$ and $\frac{d^2p(\lambda)}{d\lambda^2}_{\lambda=\lambda_*} > 0$
Calculate $N_c = n \int_0^{\lambda*} p(\lambda) d\lambda$

Algorithm 6 Learning the Graph Laplacian Moments.

Input: Normalised Laplacian L_{norm}, Number of Probe Vectors d, Number of moments required m
Output: Moments of Normalised Laplacian $\{\mu_i\}$
for i in $1, .., d$ **do**
    Initialise random vector $z_i \in R^{1 \times n}$
    for j in $1, .., m$ **do**
        $z_i' = L_{\text{norm}} z_i$
        $\rho_{ij} = z_i z_j'$
    end for
end for
$\mu_i = 1/d \times \sum_{j=1}^{d} \rho_{ij}$

3.7.1.1 Synthetic networks

The synthetic data we test consists of disconnected sub-graphs of varying sizes and cluster numbers, to which a small number of intra-cluster edges are added.

We use $d = 100$ Gaussian random vectors for our stochastic trace estimation, for both EGS and Lanczos [Ubaru et al., 2017]. We explain the procedure of going from adjacency matrix to Laplacian moments in Algorithm 6. When comparing EGS with Lanczos, we set the number of moments m equal to the number of Lanczos steps, as they are both matrix vector multiplications in

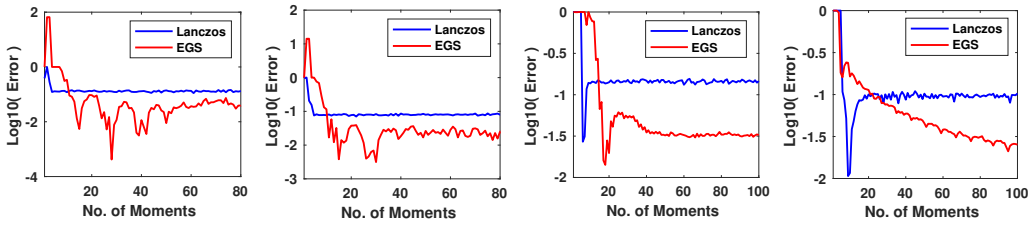
the Krylov subspace. We further use Chebyshev polynomial input instead of power moments for improved performance and conditioning. In order to normalise the moment input we use the normalised Laplacian with eigenvalues bounded by $[0, 2]$ and divide by 2. To make a fair comparison we take the output from Lanczos [Ubaru et al., 2017] and apply kernel smoothing [Lin et al., 2016] before applying our cluster number estimator.

We estimate the number of clusters and report the fractional error. The results are shown in Table 3.2. In each case, the method achieving lowest detection error is highlighted in bold. It is evident that the EGS approach outperforms Lanczos as the number of clusters and the network size increase. We observe a general improvement in performance for larger graphs, visible in the differences between fractional errors for EGS as the graph size increases and not kernel-smoothed Lanczos.

Table 3.2: Fractional error in cluster number detection for synthetic networks using EGS and Lanczos methods with 80 moments. n_c denotes the number of clusters in the network and $n = 30 \times n_c$ the number of nodes.

n_c	9	30	90	240
Lanczos	$\mathbf{3.2 \times 10^{-3}}$	1.4×10^{-2}	1.8×10^{-2}	2.89×10^{-2}
EGS	9.7×10^{-3}	$\mathbf{6.4 \times 10^{-3}}$	$\mathbf{5.8 \times 10^{-3}}$	$\mathbf{3.5 \times 10^{-3}}$

To test the performance of our approach for networks that are too large to apply eigen-decomposition, we generate two large networks by mixing the ER, WA, BA random graph models. The first large network has a size of 201,600 nodes and comprises 305 interconnected clusters whose size varies from 500 to 1000 nodes. The second large network has a size of 404,420 nodes and comprises interconnected 1355 clusters whose size varies from 200 to 400 nodes. The results in Figure 3.5 show that for both methods, the detection error generally decreases as more moments are used, and our EGS



(a) 305 clusters (b) 1,355 clusters (c) Email data (d) NetScience data

Figure 3.5: Log error of cluster number detection using EGS and Lanczos methods on large networks with (a) synthetic network with 201,600 nodes and 305 clusters and (b) synthetic network with 404,420 nodes and 1,355 clusters. (c) Email network of 1,003 nodes (d) NetScience network of 1,589 nodes.

approach again outperforms the Lanczos method for both large synthetic networks.

3.7.1.2 Small real-world networks

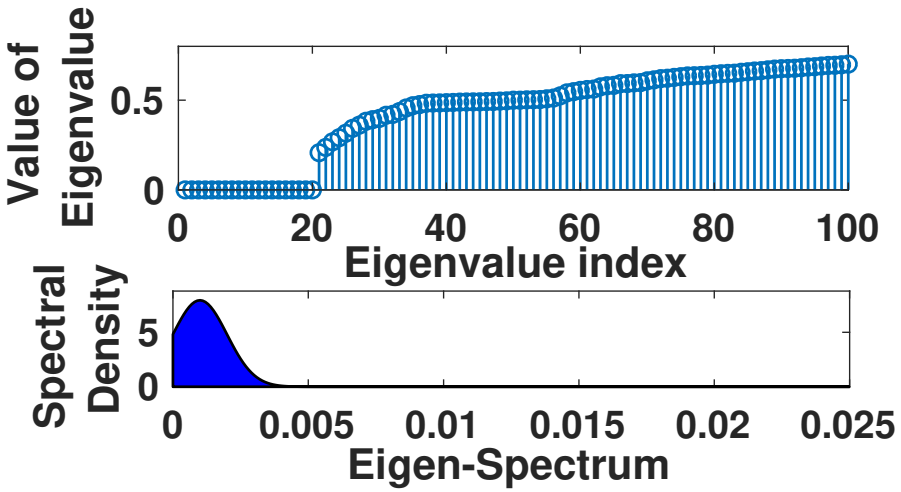


Figure 3.6: Eigenvalues of the Email dataset (shown as a stem plot up top) with clear spectral gap and $\lambda_* \approx 0.0025$, shown at the bottom of the figure as a zoom in plot. The shaded area of the spectral density multiplied by the number of nodes n predicts the number of clusters.

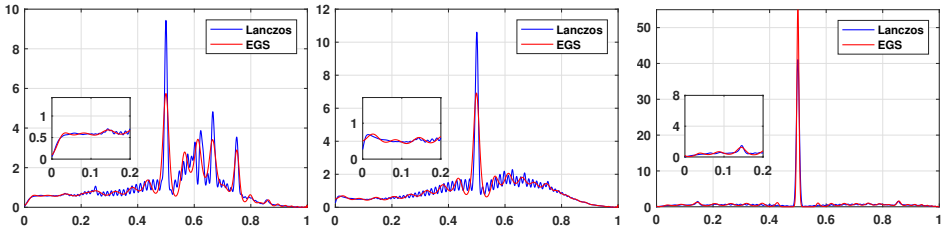
We next experiment with relatively small real-world networks, such as the

Email network in the SNAP dataset, which is an undirected graph in which the $n = 1,003$ nodes represent members of a large European research institution and the edges represent the existence of email communication between them. For such a network, we can still calculate the ground-truth number of clusters by computing the eigenvalues explicitly and finding the spectral gap near 0. For the Email network, we count 20 very small eigenvalues before a large jump in magnitude (measured in the log scale) and set this as the ground-truth. This is shown in Figure 3.6, where we display the value of each of the eigenvalues in increasing order and how this results in a broadened peak in the EGS. The area under the curve multiplied by the number of network nodes is the number of clusters n_c. We note that the number 20 differs from the value of 42 given by the number of departments at the research institute in this dataset. A likely reason for this ground-truth inflation is that certain departments, Astrophysics, Theoretical Physics and Mathematics for example, may collaborate to such an extent that their division in name may not be reflected in terms of node connection structure. We plot the log error against the number of moments for both EGS and Lanczos in Figure 3.5c, with EGS showing superior performance. We repeat the experiment on the NetScience collaboration network, which represents a co-authorship network of $1,589$ scientists ($n = 1,589$) working on network theory and experiment [Newman, 2006a]. The results in Figure 3.5d show that EGS quickly outperforms the Lanczos algorithm after around 20 moments. It is worth mentioning that the although it seems that for a certain small number of moments, changing from dataset to dataset Lanczos outperforms EGS. Outside of very small datasets, we do not have the ability to calculate the ground truth, so for a chosen smoothing coefficient, it is impossible to fine tune the number of moments so that the best result is given. Usually this moment number m is

either limited by computational capacity, or in general we would hope that for more information we would achieve more accurate results (as is the case for EGS).

3.7.1.3 Large real-world networks

For large datasets with $n \gg 10^4$, where the Cholesky decomposition becomes completely prohibitive even for powerful machines, we can no longer define a ground-truth using a complete eigen-decomposition. Alternative "ground-truths", such as regarding each set of connected components with more than 3 nodes as a community, are not universally accepted. This definition, along with that of self-declared group membership, often leads to contradictions with our definition of a community. A notable example is the Orkut dataset, where the number of stated communities is greater than the number of nodes [Leskovec and Krevl, 2014]. Beyond being impossible to learn such a value from the eigenspectra, if the main reason to learn about clusters is to partition groups and to summarise networks into smaller substructures, such a definition is undesirable. We present our findings for the number of clusters in the DBLP ($n = 317,080$), Amazon ($n = 334,863$) and YouTube



(a) DBLP (b) Amazon (c) Youtube

Figure 3.7: Approximate spectral density of two real-world networks, using smoothed Lanczos and MaxEnt, which gives a converging number of clusters with increasing number of moments m

Table 3.3: Cluster number detection by EGS for DBLP ($n = 317,080$), Amazon ($n = 334,863$) and YouTube ($n = 1,134,890$).

Moments	40	70	100
DBLP	2.215×10^4	8.468×10^3	8.313×10^3
Amazon	2.351×10^4	1.146×10^4	1.201×10^4
Youtube	4.023×10^3	1.306×10^4	1.900×10^4

($n = 1,134,890$) networks in Table 3.3. For both the DBLP and Amazon networks, the number of clusters N_c seems to converge with increasing moments number m, whereas for YouTube such a trend is not visible. This can be explained by the networks spectral desnity. For both DBLP and Amazon (Figures 3.7a and 3.7b respectively), we see that our method implies a clear spectral gap near the origin, indicating the presence of clusters. Whereas for the YouTube dataset, shown in Figure 3.7c, no such clear spectral gap is visible and hence the number of clusters cannot be estimated accurately.

3.8 Conclusion

We present an alternative to the Lanczos algorithm and kernel smoothing for the problem of spectral density estimation, applied to cluster number counting and graph similarity. The major advantage of EGS over Lanczos is the estimated densities smoothness, which for the use cases investigated here is beneficial, as we avoid an extra estimation procedure. In the next Section, where we investigate the spectral density of large neural networks and where we are specifically in individual eigenvalues which are well seperated from each other, the Lanczos algorithm, is superior. The Lanczos algorithm also gives information on the eigenvectors associated with these well seperated eigenvalues, which we use for the purposes of second-order optimisation.

4 | The Deep Visualisation Suite

In this Chapter we take the ideas of spectral visualisation of graphs (Chapter 3) using moment-matched approximations (Chapter 2) to Deep Neural Networks. The main contribution of this Chapter is a software implementation of the Lanczos algorithm on Deep Neural Network Hessians. We would like to remark that the lack of such software, which we have open-sourced to the community [1], has been a major bottleneck for curvature based neural network research, by which we mean any research that utilises the Hessian, for example second order optimisation or uncertainty calculations involving the Laplace approximation. Several largely unjustified and inferior approximations have been regularly utilised in the literature, which we detail in depth in this Chapter. The Lanczos algorithm, due to its intricate relationship with Gaussian quadrature and the problem of moments [Hausdorff, 1921], is subject to several misconceptions, even in highly cited works. Specifically it is considered to learn the largest eigenvalues in sequential order,which we explicitly disprove in this Chapter along with several showcasing experiments. Instead when seeded with a random vector, it actually learns a highly accurate moment-matched approximation to the spectral density. Whilst the moments only completely match the underlying spectral density in the limit of an infinite number of random vectors, we show both empirically and theoretically that we expect this agreement to be good for high-dimensional matrices. In the later Chapters of this thesis, we extensively utilise the software package to confirm theoretical predictions, along with questioning the validity of certain beliefs in the literature.

[1] Deep Curvature Suite Hyperlink

For the work presented in this Chapter, I provided the understanding of the Lanczos algorithm and all the proofs therein along with comparing experiments. I also provided the link between the Lanczos algorithm, the Pearlmutter trick and Gaussian quadrature/stochastic trace estimation to make effective spectral plots. Timur Garipov assisted with the implementation in Pytorch and developed the implementations of the gradient/Hessian variance and loss surface plots. Xingchen Wan extended the implementation to the generalised Gauss-Newton. I would further like to acknowledge Robin Ru for assisting me in 2017 for an earlier stochastic version of this codebase in the MATconvnet framework, which we never published. A few months after the initial version of the code was developed in Russia, Yao et al. [2018] brought out similar, although did not include Hessian variance calculation capability in their software.

4.1 Introduction

The success of Deep Neural Networks (DNNs) trained with Stochastic Gradient Descent (SGD) based optimisers on a range of tasks, has led to an explosion in the availability of easy to use out of the box high performance software implementations. Automatic differentiation packages such as TensorFlow [Abadi et al., 2016] and PyTorch [Paszke et al., 2017] have become widely adopted, with higher level packages allowing users to state their model, dataset and optimiser in a few lines of code [Chollet, 2015], effortlessly achieving state of the art performance. However, the pace of development of software extracting second-order information, representing the curvature at a point in weight space, has not kept abreast. Researchers aspiring to evaluate or use curvature information need to implement their own libraries, which are

rarely shared or kept up to date. Naive implementations are computationally intractable for all but the smallest of models. Hence, researchers typically either completely ignore curvature information or use highly optimistic approximations, such as the diagonal elements of the matrix or of a surrogate matrix, with limited justification or empirical analysis of the harshness of the aforementioned approximations Chaudhari et al. [2016], Dangel et al. [2019] The curvature of the loss informs us about the local conditioning of the problem (i.e the ratio of the largest to smallest eigenvalues $\frac{\lambda_1}{\lambda_P}$). This determines the rate of convergence for first order methods and informs us about the optimal learning and momentum rates [Nesterov, 2013]. Hence easily accessible curvature information could allow practitioners to scale their learning rates in an optimal way throughout training, instead of relying on expert scheduling, we investigate this using our software in Section 4.3.2. Research areas where curvature information features most prominently are analyses of the **Loss Surface** and **Newton** type optimization methods.

4.1.1 Loss Surfaces

Recent neural network loss surface analysis (by which we explicitly mean the Hessian, i.e the second derivative of the loss over the whole dataset) using full eigendecomposition [Sagun et al., 2016, 2018] have been limited to toy examples with less than five thousand parameters. Hence, loss surface visualisation of deep neural networks have often focused on two dimensional slices of random vectors [Li et al., 2017] or the changes in the loss traversing a set of random vectors drawn from the d-dimensional Gaussian distribution [Izmailov et al., 2018]. It is not clear that the loss surfaces of modern expressive neural networks, containing millions or billions of dimensions, can be well captured in this manner. Small experiments have shown neu-

ral networks have a large rank degeneracy [Sagun et al., 2016] with a small number of large outliers. However high dimensional concentration theorems [Vershynin, 2018] guarantee that even a large number of randomly sampled vectors are unlikely to encounter such outliers and hence have limited ability to discern the geometry between various solutions. Other works, that try to distinguish between flat and sharp minima have used the diagonal of the Fisher information matrix [Chaudhari et al., 2016], an assumption we will challenge in this chapter.

From a practical perspective, specific properties of the loss surface are not captured by the aforementioned approaches. Examples include the *flatness* as specified by the trace, Frobenius and spectral norm. These measures have been extensively used to characterise the generalisation of a solution found by SGD [Wu et al., 2018, Izmailov et al., 2018, He et al., 2019, Jastrzkbski et al., 2017, 2018, Keskar et al., 2016]. Under a Bayesian and minimum description length argument [Hochreiter and Schmidhuber, 1997] flatter minima should generalise better than sharper minima. The magnitude of these outliers have been linked to poor generalisation performance [Keskar et al., 2016] and as a consequences the generalisation benefits of large learning rate SGD [Wu et al., 2018, Jastrzkbski et al., 2017]. These properties are extremely easy in principle to estimate, at a computational cost of a small multiple of gradient evaluations. However the calculation of these properties are not typically included in standard deep learning frameworks, which limits the ability of researchers to undertake such analysis.

From a Bayesian multi-modal point of view, we have that the Evidence is given by the best fit likelihood multiplied by the Occam factor [MacKay, 1992] (p.362). The training loss is the negative log likelihood, hence the likelihood is given by $\exp\left(-L(\boldsymbol{w})\right)$. The occam factor contains a square-root

over the determinant of the Hessian (as it is a high dimensional Laplace i.e. Gaussian approximation) and hence smaller eigenvalues, i.e many eigenvalues that tend towards zero, the occam factor and hence evidence increase in size. However the training loss is also important. However for our purposes the training loss is always very small so this term is always ≈ 1. We also note that this approximation does not allow for negative or zero eigenvalues, extensively observed in practice and in this chapter. We note that Jastrzebski et al. [2020] argue that the covariance of the gradients is both close to the Hessian in terms of its largest eigenvalues and that the covariance of gradients

Other important areas of loss surface investigation include understanding the effectiveness of batch normalisation[Ioffe and Szegedy, 2015]. Recent convergence proofs [Santurkar et al., 2018] bound the maximal eigenvalue of the Hessian with respect to the activations and bounds with respect to the weights on a per layer basis. Bounds on a per layer basis do not imply anything about the bounds of the entire Hessian and furthermore it has been argued that the full spectrum must be calculated to give insights on the alteration of the landscape [Kohler et al., 2018].

4.1.2 Second Order Optimisation Methods

Second order optimisation methods solve the minimisation problem for the loss, $L(\boldsymbol{w}) \in \mathbb{R}$ associated with parameters $\boldsymbol{w} \in \mathbb{R}^{P\times 1}$ and perturbation $\boldsymbol{\delta w} \in \mathbb{R}^{P\times 1}$ to the second order in Taylor expansion,

$$\boldsymbol{\delta w}^* = \text{argmin}_{\boldsymbol{\delta w}} L(\boldsymbol{w}+\boldsymbol{\delta w}) = \text{argmin}_{\boldsymbol{\delta w}} L(\boldsymbol{w}+\boldsymbol{\delta w}) = -\bar{\boldsymbol{H}}^{-1}\nabla L(\boldsymbol{w}). \quad (4.1)$$

Sometimes, such as in deep neural networks when the Hessian $\boldsymbol{H} = \nabla^2 L(\boldsymbol{w}) \in \mathbb{R}^{P\times P}$ is not positive definite, a positive definite surrogate is used. Note that

Equation 4.1 is not lower bounded unless $\boldsymbol{H}$ is positive semi-definite. Often either a multiple of the identity is added (known as damping) to the Hessian ($\boldsymbol{H} \rightarrow \boldsymbol{H} + \gamma \boldsymbol{I}$) [Dauphin et al., 2014] or a surrogate positive definite approximation to the Hessian $\bar{\boldsymbol{H}}$, such as the Generalised Gauss-Newton (GGN) [Martens, 2010, Martens and Sutskever, 2012] is employed. To derive the GGN, we express the loss in terms of the activation $L(\boldsymbol{w}) = \sigma(f(\boldsymbol{w}))$ of the output of the final layer $f(\boldsymbol{w})$. Hence the elements of the Hessian can be written as

$$\boldsymbol{H}(\boldsymbol{w})_{ij} = \sum_{k=0}^{d_y} \sum_{l=0}^{d_y} \frac{\partial^2 \sigma(f(\boldsymbol{w}))}{\partial f_l(\boldsymbol{w}) \partial f_k(\boldsymbol{w})} \frac{\partial f_l(\boldsymbol{w})}{\partial w_j} \frac{\partial f_k(\boldsymbol{w})}{\partial w_i} + \sum_{k=0}^{d_y} \frac{\partial \sigma(f(\boldsymbol{w}))}{\partial w_j} \frac{\partial^2 f_k(\boldsymbol{w})}{\partial w_j \partial w_i}. \tag{4.2}$$

The first term on the LHS of Equation 5.23 is known as the *Generalised Gauss-Newton* matrix.

Despite the success of second order optimisation methods using the GGN for difficult problems on which SGD is known to stall, such as recurrent neural networks [Martens and Sutskever, 2012], or auto-encoders [Martens, 2016]. Researchers wanting to implement second order methods such as [Vinyals and Povey, 2012, Martens and Sutskever, 2012, Dauphin et al., 2014] face the aforementioned problems of difficult implementation. As a small side note, Bayesian neural networks using the Laplace approximation feature the Hessian inverse multiplied by a vector [Bishop, 2006].

Eigendecomposition: Whilst we motivate the importance of the Hessian, we note that due to the spectral decomposition theorem, the Hessian is uniquely defined by its eigenvalue/eigenvector pairs. Furthermore, we may often only be interested in a small number of eigenvectors (the outliers usually) and the density of the eigenvalues λ as a function of λ, hence for much

of our work, inspecting the spectral density $p(\lambda)$ is key as opposed to the Hessian itself.

4.1.3 Contributions

We introduce our package, the **Deep Curvature suite**, a software package that allows analysis and visualisation of deep neural network curvature. The main features and functionalities of our package are:

- **Eigenspectrum analysis of the curvature matrices** Powered by the Lanczos [Meurant and Strakoš, 2006] techniques implemented in GPyTorch [Gardner et al., 2018] and outlined in Section 3, *with a single random vector* we use the Pearlmutter matrix-vector product trick [Pearlmutter, 1994] for fast inference of the eigenvalues and eigenvectors of the common curvature matrices of the deep neural networks. In addition to the standard Hessian matrix, we also include the feature for inference of the eigen-information of the Generalised Gauss-Newton matrix, a commonly used positive-definite surrogate to Hessian[2].

- **Advanced Statistics of Networks** In addition to the commonly used statistics to evaluate network training and performance such as the training and testing losses and accuracy, we support computations of more advanced statistics: For example, we support squared mean and variance of gradients and Hessians (and GGN), squared norms of Hessian and GGN, L2 and L-inf norms of the network weights and etc. These statistics are useful and relevant for a wide range of purposes

[2]The computation of the GGN-vector product is similar with the computational cost of two backward passes in the network. Also, GGN uses *forward-mode automatic differentiation* (FMAD) in addition to the commonly employed *backward-mode automatic differentiation* (RMAD). In the current PyTorch framework, the FMAD operation can be achieved using two equivalent RMAD operations.

such as the designs of second-order optimisers and network architecture.

- **Visualisations** For all main features above, we include accompanying visualisation tools. In addition, with the eigen-information obtained, we also feature visualisations of the **loss landscape** by studying the sensitivity of the neural network to perturbations of weights.
- **Second Order Optimisation** We include out of the box stochastic lanczos second order optimisers which use the absolute Hessian [Dauphin et al., 2014] or the Generalised Gauss Newton [Martens, 2010] as curvature matrices
- **Batch Normalisation Support** Our code allows for the use of batch normalisation[Ioffe and Szegedy, 2015] in neural networks, which is integrated by default on most modern convolutional neural networks. Since the way in watch batch normalisation is calculated has a big effect on the resulting curvature calculations and can be dependent on the order in which the samples are processed, we offer two models of evaluation. *Train mode* which depends on the order in which the samples are processed and is similar to what the optimiser sees during training and *evaluation mode*, in which the statistics are pre-computed and hence invariant to sample shuffling.

Our software makes calculating and visualising curvature information as simple as calculating the gradient at a saved checkpoint in weight-space. The computational complexity of our approach is $\mathcal{O}(mP)$, where m is the number of Lanczos steps [3] and P is the number of model parameters. This in stark

[3] which corresponds to the number of moments of the underlying Hessian/GGN density which we wish to match

Functionality	Deep Curvature Suite	PyHessian
Hessian Density Estimation	✓	✓
GGN Density Estimation	✓	✗
Loss Surface Visualisation	✓	✓
Gradient Variance	✓	✗
Hessian/GGN Variance	✓	✗
Second Order Optimisers	✓	✗

Figure 4.1: Table of Contributions: comparison between our released software and that of another PyTorch based Hessian tool PyHessian

contrast full exact eigendecomposition which has a numerical cost of $\mathcal{O}(P^3)$, which is infeasible for large neural networks.

4.1.4 Related Work

Recent work making curvature information more available using diagonal approximations, disallows the use of batch normalisation[Dangel et al., 2019]. Our software extends seamlessly to support batch normalization. Independent work has also uses the Lanczos algorithm for Hessian computation Yao et al. [2019], Ghorbani et al. [2019], Papyan [2019]. We compare the functionality of our package to that of PyHessian [Yao et al., 2018] in Table 1.

One major extra functionality we offer is the calculation of the GGN. For common activatios and loss functions, such as the cross-entropy loss and sigmoid activation, the generalised Gauss-Newton is equivalent to the Fisher information matrix [Pascanu and Bengio, 2014]. Hence this approximation to the Hessian is interesting in its own right and many theoretical analyses, consider the generalised Gauss-Newton or Fisher information matrices as opposed to the Hessian [Pennington and Worah, 2018, pap]. Hence accessing the GGN spectrum could be a major asset to researchers. Another extra

functionality we offer is the calculation of the Hessian/GGN variance. This quantity has been used to derive optimal learning rates in Wu et al. [2018] and to predict the effect of minibatching on curvature using a random matrix theory model, which we introduce in Chapter 5.

4.2 An Illustrated Example

We give an illustration on an example of using the Deep Curvature package. We train a VGG16 network on CIFAR-100 for 100 epochs. With the checkpoints generated, we may now compute analyse the eigenspectrum of the curvature matrix evaluated at the desired point in training.To evaluate the Hessian/GGN with 100 Lanczos iterations, we run the code given in Table 4.1.

Network Training

```
train_network(
dir='VGG16/',
dataset='CIFAR100',
data_path='data/',
epochs=100, model='VGG16',
optimizer='SGD',
optimizer_kwargs=
{'lr': 0.03, 'mom': 0.9,
'weight_decay': 5e-4})
```

Eigenspectrum Computation

```
compute_eigenspectrum(
dataset='CIFAR100',
data_path='data/',
model='VGG16',
checkpoint_path='checkpoint.pt',
save_spectrum_path='spectrum-ggn',
save_eigvec=True,
lanczos_iters=100,
curvature_matrix='ggn_lanczos',)
```

Table 4.1: Training a Neural Network and Calculating the Eigenspectrum using the Deep Curvature Suite package

This function call saves the spectrum results (including eigenvalues, eigenvectors and other related statistics) in the `save_spectrum_path` path string defined. To visualise the spectrum as stem plot we simply run the code given in Table 4.2 with corresponding example Figure shown.

Eigenspectrum Visualization

```
plot_spectrum('lanczos',
path='spectrum-ggn.npz')
plt.show()
```

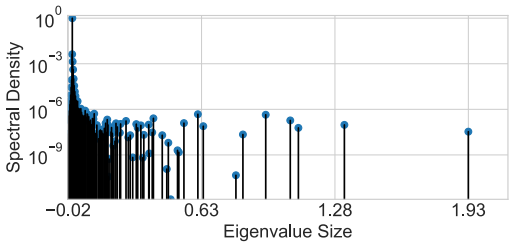


Table 4.2: Eigenspectrum plotting code and corresponding stem plot

Finally, with the eigenvalues and eigenvectors computed, we might be interested in knowing how sensitive the network is to perturbation along these directions. To achieve this, we first construct a loss landscape by setting the number of query points and maximum perturbation to apply. To achieve that, we call the code given in Table 4.3. In this example, we set the maxi-

Loss Surface

```
build_landscape(
dataset='CIFAR100',
data_path='data/',
model='VGG16',
dist=1., n=21,
spec_path=hessian',
ckpt_path='ck.pt',
sv_path='scape.npz'
plot_landscape
('landscape.npz')
plt.show())
```

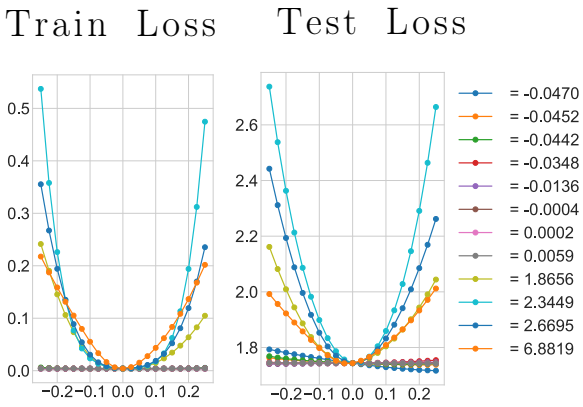


Table 4.3: Loss surface visualization along the sharpest Hessian eigenvectors with the Deep Curvature Suite. Note the similarity between the test and train curves for the eigenvectors corresponding to the sharpest eigenvalues, also note that flat in the train loss surface is not necessarily flat in the test loss surface.

mum perturbation to be 1 (`dist` argument) and number of query points along each direction to be 21 (`n_points` argument). The corresponding plots in Training and Testing loss are also shown.

4.3 Example Research Applications

In this section we list 3 example scientific applications of our software package. We investigate second order optimisation for deep neural networks using our inbuild stochastic Lanczos optimiser, online learning rate scheduling and verifying predictions made using random matrix theory in theoretical deep learning.

4.3.1 Simplicity of Second Order Optimisation with the Deep Curvature Suite

To enable researchers to experiment with stochastic second order optimisation algorithms for deep neural networks, we implement a Lanczos based optimiser (which takes the absolute Hessian Dauphin et al. [2014], or Generalised Gauss Newton Martens [2010] as input). The code for running this optimiser is summarised in Table 4.4. We plot the training error of the VGG-

SGD Training

```
train_network(
dir='VGG16-CIFAR100/',
dataset='CIFAR100',
data_path='data/',epochs=100,
model='VGG16', optimizer='SGD',
optimizer_kwargs={
'lr': 0.01,
'momentum': 0.9,
'weight_decay': 0
'batch_size': 128})
```

Lanczos Training

```
train_network(
dir='VGG16-CIFAR100/',
dataset='CIFAR100',
data_path='data/',epochs=100,
model='VGG16', optimizer='LancGN',
optimizer_kwargs={
'lr': 1,
'damping': 10,
'weight_decay': 0
'batch_size': 128
'curvature_batch_size': 128})
```

Table 4.4: Comparison of SGD training and Second Order Optimisation using the Deep Curvature Suite

16 network on CIFAR-100 dataset, which has over 16 million parameters and hence is out of reach of full inversion methods, in Figure 4.2b. We keep the ratio of damping constant to learning rate constant, where $\delta = 10\alpha$, for a variety of learning rates in $\{1, 0.1, 0.01, 0.001, 0.0001\}$ with a batch size of 128 for both the gradient and the curvature, all of which post almost identical performance. We also compare against different learning rates of Adam and the best grid searched learning rate of SGD, both of which converge significantly slower per iteration compared to our stochastic Newton methods. We note that in terms of practical optimisation that the cost of running a stochastic second order optimiser is far greater than that of SGD. For this example, where we use 20 Lanczos iterations, the per iteration cost scales as ≈ 40 times that of SGD. Hence we expect these algorithms simply to serve as useful baselines against more computationally efficient approximations. One example future direction could be to invert the matrices less frequently as done in Martens and Grosse [2015], reducing computational cost.

4.3.2 Spectral Stochastic Gradient Descent with the Deep Curvature Suite

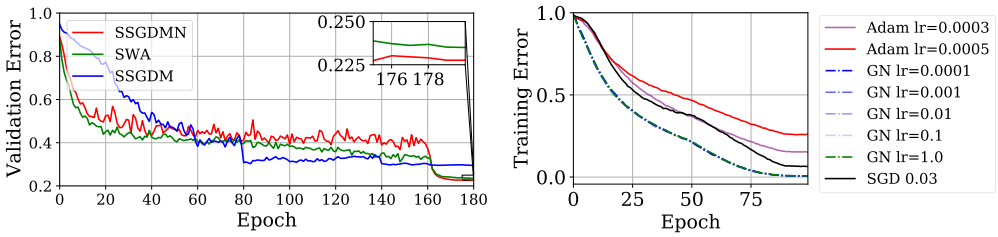
SGD is incredibly sensitive to the choice of initial learning rate and its scheduling. Whilst adaptive methods are considered more robust to the initial learning rate choice and its schedule, they have been shown to perform poorly on validation or held out test data [Wilson et al., 2017]. Hence practitioners typically experiment with a variety of schedules, such as the step, linear, exponential, polynomial and cosine annealing. One potential application of our codebase is to use a stochastic estimate of the curvature and to use established optimal learning and momentum rates for the local quadratic approximation at the given point in weightspace [Nesterov, 2013],

given in Equation 4.3.

$$\alpha_{Polyak} = \frac{2}{\sqrt{\lambda_1} + \sqrt{\lambda_P}}, \rho = \left(\frac{\sqrt{\lambda_1} - \sqrt{\lambda_P}}{\sqrt{\lambda_1} + \sqrt{\lambda_P}}\right)^2, \alpha_{Nesterov} = \sqrt{\frac{\lambda_P}{\lambda_1}}\rho = \frac{\sqrt{\lambda_1} - \sqrt{\lambda_P}}{\sqrt{\lambda_1} + \sqrt{\lambda_P}}. \tag{4.3}$$

Where λ_1, λ_P refer to the largest and smallest eigenvalues of the Hessian respectively. The learning rates and momenta α, ρ are given for both Polyak and Nesterov momentum respectively. There are several complications to the above method. Firstly the Hessian of neural networks is in general not positive definite and hence $\lambda_P < 0$ (these optimality formulae are strictly for convex methods). Secondly the loss surface at each point in weight space is likely to be different and it is impractical to recalculate the optimal step size using Equation 4.3 for every iteration, even if we sub-sample the data. From the work we develop in chapter 5, that the loss surface when using a mini-batch varies to that of the full surface (it is sharper) and hence the learning rates when training using a mini-batch must be optimal for the minibatch surface and not that of the full dataset. As an example use case for our software we consider whether the method of Izmailov et al. [2018], which combines SGD with Polyak averaging [Polyak and Juditsky, 1992] at the end of training (which they denote *SWA*), to improve generalisation performance, could be improved. Instead of fine tuning the linear decay learning rate schedule by hand, we instead learn the learning rate using Equation 4.3 at regular intervals by estimating the curvature using a sub-sampled Lanczos spectral estimate.

We run the preactivated ResNet-110 on the CIFAR-100 dataset for 180 epochs and learn the Polyak and Nesterov learning rates and momenta every 20 epochs using the same mini-batch size as the gradient batch size 128 for 20 Lanczos iterations. Since the smallest Ritz values are very very close to



(a) Test Error CIIFAR-100 SSGDM(N) & SWA on the PreResNet-110

(b) Train Error VGG-16 CIFAR-100 Second Order Optimisers

0, which would result in a momentum $\rho = 1$, we use a heuristic to remove the smallest Ritz values, whereby if the Ritz value of largest spectral mass has more than 50% of the spectral mass, it is removed and the resulting density renormalised, forming the new spectral density of interest. All methods employ Polyak Avearging at epoch 161 as in Izmailov et al. [2018]. We plot the results in Figure 4.2a and show that the Nesterov variant (SSGDMN) of online learning rate and momentum learning performs well against the baseline (SWA). Interestingly the Polyak variant (SSGDM) learns to drop the learning rate at regular intervals, similar to classical training methods, which does not perform as well as keeping the learning rate high and using Polyak averaging at the end of training.

4.4 Lanczos Primer

The Lanczos Algorithm (Algorithm 7), for which we use the GPU implementation [Gardner et al., 2018], is an iterative algorithm for learning a subset of the eigenvalues/eigenvectors of any Hermitian matrix, requiring only matrix vector products. It can be regarded as a far superior adaptation of the power iteration method, where the Krylov subspace $\mathcal{K}(\boldsymbol{H}, \boldsymbol{v}) = \text{span}\{\boldsymbol{v}, \boldsymbol{H}^2\boldsymbol{v}, \boldsymbol{H}^3\boldsymbol{v}...\}$ is orthogonalised using Gram-Schmidt procedure. Beyond having improved convergence compared to the power iteration method

[Bai et al., 1996] by storing the intermediate orthogonal vectors in the corresponding Krylov subspace, the Lanczos approach produces estimates of the eigenvectors and eigenvalues of smaller absolute magnitude, known as the Ritz vectors/values. Despite its known superiority to the power iteration method and relationship to orthogonal polynomials (hence when combined with random vectors the ability to estimate the entire spectrum of a matrix). These properties are often ignored or forgotten by practitioners, so we detail them here.

4.4.1 Comparison with Power Iterations

The Lanczos method be be explicitly derived by considering the optimisation of the Rayleigh quotient [Golub and Van Loan, 2012], defined as

$$r(\boldsymbol{v}) = \frac{\boldsymbol{v}^T \boldsymbol{H} \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{v}} \tag{4.4}$$

over the entire Krylov subspace $\mathcal{K}_m(\boldsymbol{H}, \boldsymbol{v})$ where $\boldsymbol{v}$ is a vector, as opposed to power iteration which is a particular vector in the Krylov subspace $\boldsymbol{u} = \boldsymbol{H}^m \boldsymbol{v}$. Despite this, practitioners looking to learn the leading eigenvalue very often resort to the power iteration, likely due to its implementational simplicity. We showcase the methods relative inferiority to Lanczos with the following convergence theorems

Theorem 4. *Let $\boldsymbol{H}^{P \times P}$ be a symmetric matrix with eigenvalues $\lambda_1 \geq .. \geq \lambda_P$ and corresponding orthonormal eigenvectors $z_1, ..z_P$. If $\theta_1 \geq .. \geq \theta_m$ are the eigenvalues of the matrix T_m obtained after m Lanczos steps and $q_1, ...q_m$ the*

corresponding Ritz eigenvectors then

$$\lambda_1 \geq \theta_1 \geq \lambda_1 - \frac{(\lambda_1 - \lambda_n)\tan^2(\theta_1)}{(c_{m-1}(1+2\rho_1))^2},\ \lambda_P \leq \theta_k \leq \lambda_m + \frac{(\lambda_1 - \lambda_n)\tan^2(\theta_1)}{(c_{m-1}(1+2\rho_1))^2} \tag{4.5}$$

where c_k *is the Chebyshev polyomial of order* k. $\cos\theta_1 = |\boldsymbol{q}_1^T \boldsymbol{z}_1|$ *&* $\rho_1 = (\lambda_1 - \lambda_2)/(\lambda_2 - \lambda_n)$

Proof. see [Golub and Van Loan, 2012]. □

Theorem 5. *Assuming the same notation as in Theorem 4, after* m *power iteration steps the corresponding extremal eigenvalue estimate is lower bounded by*

$$\lambda_1 \geq \theta_1 \geq \lambda_1 - (\lambda_1 - \lambda_n)\tan^2(\theta_1)\left(\frac{\lambda_2}{\lambda_1}\right)^{2m-1} \tag{4.6}$$

From the rapid growth of orthogonal polynomials such as Chebyshev, we expect Lanczos to be superior for larger spectral gap and iteration number. To verify this experimentally, we collect the non-identical terms in the Equations 4.5 and 4.6 for the lower bounds of λ_1, derived by Lanczos and Power iteration and denote them L_{k-1} and R_{k-1} respectively. For different values of λ_1/λ_2 and iteration number m we give the ratio of these two quantities in Table 4.5. As can be clearly seen, the Lanczos lower bound is always closer to the true value. This improves with the iteration number m and its relative edge is reduced if the spectral gap is decreased.

λ_1/λ_2	$m=5$	$m=10$	$m=15$	$m=20$
1.5	$\frac{1.1\times10^{-4}}{3.9\times10^{-2}}$	$\frac{2\times10^{-10}}{6.8\times10^{-4}}$	$\frac{3.9\times10^{-16}}{1.2\times10^{-5}}$	$\frac{7.4\times10^{-22}}{2.0\times10^{-7}}$
1.1	$\frac{2.7\times10^{-2}}{4.7\times10^{-1}}$	$\frac{5.5\times10^{-5}}{1.8\times10^{-1}}$	$\frac{1.1\times10^{-7}}{6.9\times10^{-2}}$	$\frac{2.1\times10^{-10}}{2.7\times10^{-2}}$
1.01	$\frac{5.6\times10^{-1}}{9.2\times10^{-1}}$	$\frac{1.0\times10^{-1}}{8.4\times10^{-1}}$	$\frac{1.5\times10^{-2}}{7.6\times10^{-1}}$	$\frac{2.0\times10^{-3}}{6.9\times10^{-1}}$

Table 4.5: L_{k-1}/R_{k-1} For different values of spectral gap λ_1/λ_2 and iteration number m, Table from [Golub and Van Loan, 2012]

4.4.2 The Problem of Moments: Spectral Density Estimation Using Lanczos

In this section we show that that the Lanczos Tri-Diagonal matrix corresponds to an orthogonal polynomial basis which matches the moments of $\boldsymbol{v}^T\boldsymbol{H}^m\boldsymbol{v}$ and that when $\boldsymbol{v}$ is a zero mean random vector with unit variance, these moments correspond to the moments of the underlying spectral density $p(\lambda)$. *Stochastic trace estimation*, which we detail in Section 2.5 shows that for zero mean, unit variance random vectors, the relationship

$$\mathbb{E}_{\boldsymbol{v}}\mathrm{Tr}\Big((\boldsymbol{v}^T\boldsymbol{H}^m\boldsymbol{v})\Big) = \mathrm{Tr}\Big(\mathbb{E}_{\boldsymbol{v}}(\boldsymbol{v}\boldsymbol{v}^T\boldsymbol{H}^m)\Big) = \mathrm{Tr}(\boldsymbol{H}^m) = \sum_{i=1}^{P}\lambda_i^m = P\int_{\lambda\in\mathcal{D}}\lambda^m d\mu(\lambda)$$

holds, where $\mathcal{D}$ is the domain of the eigenvalues and $\mu(\lambda)$ is a measure. Where we have used the linearity of trace and expectation. Hence in expectation over the set of random vectors, the trace of the inner product of $\boldsymbol{v}$ and $\boldsymbol{H}^m\boldsymbol{v}$ is equal to the m'th moment of the spectral density of $\boldsymbol{H}$. For specific matrices for which we expect the condition number to be bounded asymptotically, we expect a concentration result to hold asymptotically in the matrix dimension P for even a single vector.

Lanczos-Stieltjes The Lanczos tri-diagonal matrix $\boldsymbol{T}$ can be derived from the Moment matrix $\boldsymbol{M}$, corresponding to the discrete measure $d\mu(\lambda)$ satisfying the moments $\mu_i = \boldsymbol{v}^T\boldsymbol{H}^i\boldsymbol{v} = \int \lambda^i d\mu(\lambda)$ [Golub and Meurant, 1994]

$$\boldsymbol{M} = \begin{bmatrix} 1 & \boldsymbol{v}^T\boldsymbol{H}\boldsymbol{v} & \dots & \boldsymbol{v}^T\boldsymbol{H}^{m-1}\boldsymbol{v} \\ \boldsymbol{v}^T\boldsymbol{H}\boldsymbol{v} & \boldsymbol{v}^T\boldsymbol{H}^2\boldsymbol{v} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \boldsymbol{v}^T\boldsymbol{H}^{m-1}\boldsymbol{v} & \dots & \dots & \boldsymbol{v}^T\boldsymbol{H}^{2m-2}\boldsymbol{v} \end{bmatrix}$$

Hence for a zero mean unit variance initial seed vector, the eigenvector/eigenvalue pairs of $\boldsymbol{T}$ contain information about the spectral density of $\boldsymbol{H}$ as shown in Section 2.5. This is given by the following Theorem:

Theorem 6. *The eigenvalues of T_k are the nodes t_j of the Gauss quadrature rule, the weights w_j are the squares of the first elements of the normalised eigenvectors of T_k*

Proof. See Golub and Meurant [1994] □

A quadrature rule is a relation of the form,

$$\int_a^b f(\lambda)d\mu(\lambda) = \sum_{j=1}^{M} \rho_j f(t_j) + R[f] \tag{4.7}$$

for a function f, such that its Riemann-Stieltjes integral and all the moments exist on the measure $d\mu(\lambda)$, in the interval $[a, b]$ where $R[f]$ denotes the unknown remainder. ρ_j are the weights for the corresponding summand. The first term on the RHS of Equation 4.7 (using Theorem 6) can be seen as a discrete approximation to the spectral density matching the first m moments $\boldsymbol{v}^T\boldsymbol{H}^m\boldsymbol{v}$ [Golub and Meurant, 1994, Golub and Van Loan, 2012]

For n_v starting vectors, the corresponding discrete spectral density is given as

$$p(\lambda) = \frac{1}{n_v} \sum_{l=1}^{n_v} \left(\sum_{k=1}^{m} (\tau_k^{(l)})^2 \delta(\lambda - \lambda_k^{(l)}) \right), \tag{4.8}$$

where $\tau_k^{(l)}$ corresponds to the first entry of the eigenvector of the k-th eigenvalue, λ_k, of the Lanczos tri-diagonal matrix, $\boldsymbol{T}$, for the l-th starting vector [Ubaru and Saad, Lin et al., 2016].

4.4.3 Computational Complexity

For large matrices, the computational complexity of the algorithm depends on the Hessian vector product, which for neural networks is $\mathcal{O}(mNP)$ where P denotes the number of parameters in the network, m is the number of Lanczos iterations and N is the number of data-points. The full re-orthogonalisation adds two matrix vector products, of cost $\mathcal{O}(m^2 P)$, where typically $m^2 \ll N$. Each random vector used can be seen as another full run of the Lanczos algorithm, so for d random vectors the total complexity is $\mathcal{O}(dmP(N+m))$

Importance of keeping orthogonality: The update equations of the Lanczos algorithm lead to a tri-diagonal matrix $\boldsymbol{T} = \mathbb{R}^{m \times m}$, whose eigenvalues represent the approximated eigenvalues of the matrix $\boldsymbol{H}$ and whose eigenvectors, when projected back into the the Krylov-subspace, $\mathscr{K}(\boldsymbol{H}, \boldsymbol{v})$, give the approximated eigenvectors of $\boldsymbol{H}$. In finite precision, it is known [Meurant and Strakoš, 2006] that the Lanczos algorithm fails to maintain orthogonality between its Ritz vectors, with corresponding convergence failure. In order to remedy this, we re-orthonormalise at each step [Bai et al., 1996] (as shown in line 9 of Algorithm 7) and observe a high degree of orthonormality between the Ritz eigenvectors. Orthonormality is also essential

Algorithm 7 Lanczos Algorithm

Input: Hessian vector product $\{\boldsymbol{H}\boldsymbol{v}\}$, number of steps m
Output: Ritz eigenvalue/eigenvector pairs $\{\lambda_i, \boldsymbol{u}_i\}$ & quadrature weights τ_i
Set $\boldsymbol{v} := \boldsymbol{v}/\sqrt{(\boldsymbol{v}^T\boldsymbol{v})}$
Set $\beta := 0$, $\boldsymbol{v}_{old} := \boldsymbol{v}$
Set $\boldsymbol{V}(:, 1) := \boldsymbol{v}$
for j in $1, .., m$ **do**
    $\boldsymbol{w} = \boldsymbol{H}\boldsymbol{v} - \beta\boldsymbol{v}_{old}$
    $\boldsymbol{T}(j, j) = \alpha = \boldsymbol{w}^T\boldsymbol{v}$
    $\boldsymbol{w} = \boldsymbol{w} - \alpha\boldsymbol{w} - \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{w}$
    $\beta = \sqrt{\boldsymbol{w}^T\boldsymbol{w}}$
    $\boldsymbol{v}_{old} = \boldsymbol{v}$
    $\boldsymbol{v} = \boldsymbol{w}/\beta$
    $\boldsymbol{V}(:, j + 1) = \boldsymbol{v}$
    $\boldsymbol{T}(j, j + 1) = \boldsymbol{T}(j + 1, 1) = \beta$
end for
$\{\lambda_i, \boldsymbol{e}_i\} = eig(\boldsymbol{T})$
$\boldsymbol{u}_i = \boldsymbol{V}\boldsymbol{e}_i$
$\tau_i = (\boldsymbol{e}_i^T[1, 0, 0...0])^2$

for achieving accurate spectral resolution as the Ritz value weights are given by the squares of the first elements of the normalised eigenvectors. For the practitioner wishing to reduce the computational cost of maintaining orthogonality, there exist more elaborate schemes [Meurant and Strakoš, 2006, Golub and Meurant, 1994].

4.4.4 Common Myths in the Literature

- We can learn the negative and interior eigenvalues by shifting and inverting the matrix sign $\boldsymbol{H} \rightarrow -\boldsymbol{H} + \mu\boldsymbol{I}$ [Maddox et al., 2019] [4]. Where $\mu \in \mathbb{R}$ is some number.

[4] As can be seen in the codebase of the Maddox et al. [2019] paper link given by the comment *If the largest eigenvalue is positive, shift matrix so that any negative eigenvalue is now the largest*

- Lanczos learns the largest m $[\lambda_i, \boldsymbol{u}_i]$ pairs of $\boldsymbol{H} \in \mathbb{R}^{P\times P}$ with high probability [Dauphin et al., 2014]

Since these two related beliefs are prevalent, beyond the two citations given here, we disprove them explicitly in this section with Theorems 7 and 8.

Theorem 7. *The shift and invert procedure* $\boldsymbol{H} \rightarrow -\boldsymbol{H} + \mu\boldsymbol{I}$*, changes the Eigenvalues of the Tri-diagonal matrix* $\boldsymbol{T}$ *(and hence the Ritz values) to* $\lambda_i = -\lambda_i + \mu$

Proof. Following the equations from Algorithm 7

$$\begin{aligned}
&\boldsymbol{w}_1^T = (-\boldsymbol{H} + \mu\boldsymbol{I})\boldsymbol{v}_1 \ \&\alpha_1 = \boldsymbol{v}_1^T\boldsymbol{H}\boldsymbol{v}_1 + \mu\boldsymbol{I} \\
&\boldsymbol{w}_2 = \boldsymbol{w}_1 - \alpha_1\boldsymbol{v}_1 = (\boldsymbol{H} + \mu\boldsymbol{I})\boldsymbol{v}_1 - (\boldsymbol{v}_1^T\boldsymbol{H}\boldsymbol{v}_1 + \mu\boldsymbol{I})\boldsymbol{v}_1 \\
&\boldsymbol{w}_2 = (\boldsymbol{H} - \boldsymbol{v}_1^T\boldsymbol{H}\boldsymbol{v}_1)\boldsymbol{v}_1 \ \&\boldsymbol{v}_2 = \boldsymbol{w}_2/||\boldsymbol{w}_2|| \\
&\alpha_2 = \boldsymbol{v}_2^T(-\boldsymbol{H} + \mu\boldsymbol{I})\boldsymbol{v}_2 = -\boldsymbol{v}_2^T\boldsymbol{H}\boldsymbol{v}_2 + \mu, \beta_2 = ||\boldsymbol{w}_2||
\end{aligned} \tag{4.9}$$

assuming for $m-1$ and repeating the above steps for m we prove by induction and finally arrive at the modified Tridiagonal Lanczos matrix $\tilde{\boldsymbol{T}}$,

$$\begin{aligned}
&\tilde{\boldsymbol{T}} = -\boldsymbol{T} + \mu\boldsymbol{I} \\
&\tilde{\lambda_i} = -\lambda_i + \mu \text{ for} 1 \leq i \leq m
\end{aligned} \tag{4.10}$$

□

Remark. *We observe that no new Eigenvalues of the matrix* $\boldsymbol{H}$ *are learned. Although it is obvious that the addition of the identity does not change the Krylov subspace, such procedures are common. This disproves the first myth.*

Theorem 8. *For any matrix* $\boldsymbol{H} \in \mathbb{R}^{P\times P}$ *such that* $\lambda_1 > \lambda_2 ..... > \lambda_P$ *and* $\sum_{i=1}^{m}\lambda_i < \sum_{i=m+1}^{P}\lambda_i$*, in expectation over the set of random vectors* $\boldsymbol{v}$ *the*

m eigenvalues of the Lanczos Tridiagonal matrix $\boldsymbol{T}$ do not correspond to the top m eigenvalues of $\boldsymbol{H}$

Proof. Let us consider the matrix $\tilde{\boldsymbol{H}} = \boldsymbol{H} - \frac{\lambda_{m+1}+\lambda_m}{2}\boldsymbol{I}$.

$$\begin{cases} \lambda_i > 0, & \forall i \leq m \\ \lambda_i < 0, & \forall i > m \end{cases} \tag{4.11}$$

Under the assumptions of the theorem, $\mathrm{Tr}(\tilde{\boldsymbol{H}}) < 0$ and hence by Theorem 6 and Equation 4.7 there exist no $w_i > 0$ such that

$$\sum_{i=1}^{m} w_i \lambda_i^k = \frac{1}{P} \sum_{i=1}^{P} \lambda_i^k \forall \quad 1 \leq k \leq m \tag{4.12}$$

is satisfied for $k = 1$ As the L.H.S is manifestly positive and the RHS is negative. By Theorem 7 this holds for the original matrix $\boldsymbol{H}$. □

Remark. *Given that Theorem 8 is satisfied over the expectation of the set of random vectors, which by the Central Limit Theorem is realised as the number of random vectors $d \to \infty$, the only way to really span the top m eigenvectors is to have selected a vector which lies in the m dimensional subspace of the P dimensional problem corresponding to those vectors. This would correspond to knowing those vectors a priori, defeating the point of using Lanczos at all.*

Another way to see this, is through Theorem 4, which gives a bound on the distance between the smallest Lanczos Ritz value and the minimal eigenvalue. Intuitively, as the Ritz values and weights form a discrete m-moment spectral approximation to the Hessian spectrum, so the support of the discrete density, cannot approximately match the largest m eigenvalues. This can be seen in Figure 4.3c where we run Lanczos with $m = 30$ steps and capture

the shape of the spectral density of a $\boldsymbol{H} \in \mathbb{R}^{10000\times 10000}$, matrix including the negative eigenvalue of largest magnitude.

4.5 Synthetic Experiments

In this section, we use some examples on small random matrices to showcase the power of the Lanczos algorithm with random vectors to learn the spectral density. We compare to both the ground truth and commonly used diagonal approximations. Here, we look at known random matrices with elements drawn from specific distributions which converge to known spectral densities in the asymptotic limit. We consider both the *Wigner Ensemble* [Wigner, 1993] and the *Marchenko-Pastur* [Marchenko and Pastur, 1967], both of which are extensively used in simulations or theoretical analyses of deep neural network spectra [Pennington and Bahri, 2017, Choromanska et al., 2015a, Anonymous, 2020]. We keep the matrices small so they can be solved by exact eigendecomposition to give a ground truth.

Wigner matrices: are defined in Definition 8.1. Their distributions of eigenvalues are governed by the semi-circle distribution law (Theorem 9).

Definition 8.1. *Let* $\{Y_i\}$ *and* $\{Z_{ij}\}_{1\leq i\leq j}$ *be two real-valued families of zero mean, i.i.d. random variables, Furthermore suppose that* $\mathbb{E}|Z_{12}|^2 = 1$ *and for each* $k \in \mathbb{N}$

$$\max(\mathbb{E}|Z_{12}|^k, \mathbb{E}|Y_1|^k) < \infty \tag{4.13}$$

Consider a $P \times P$ *symmetric matrix* M_P*, whose entries are given by*

$$\begin{cases} M_P(i,i) = Y_i \\ M_P(i,j) = Z_{ij} = M_P(j,i), \end{cases} \tag{4.14}$$

The Matrix M_P is known as a real symmetric Wigner matrix.

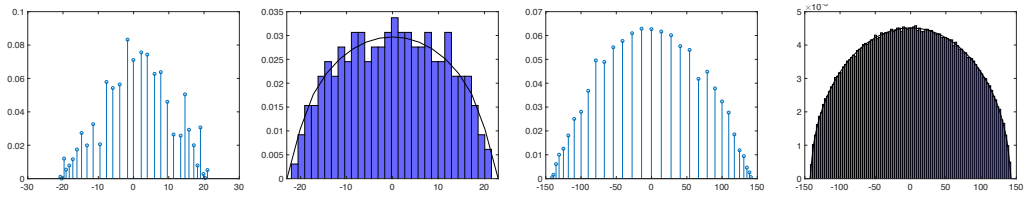
Theorem 9. *Let $\{M_P\}_{P=1}^{\infty}$ be a sequence of Wigner matrices, and for each P denote $X_P = M_P/\sqrt{P}$. Then μ_{X_P} converges weakly, almost surely to the semi circle distribution,*

$$p(\lambda) = \frac{1}{2\pi}\sqrt{4-\lambda^2}\mathbf{1}_{|\lambda|\leq 2} \tag{4.15}$$

For our experiments, we generate random matrices $\boldsymbol{H} \in \mathbb{R}^{P\times P}$ with elements drawn from the distribution $\mathcal{N}(0,1)$ for $P = \{225, 10000\}$ and plot histograms of the spectra found by eigendecomposition, along with the predicted Wigner density (scaled by a factor of $\sqrt{P}$) in Figures 4.3b & 4.3d. We compare them to the discrete spectral density approximation learned by lanczos in $m = 30$ steps using a single random vector $d = 1$ in Figures 4.3a & 4.3c. It can be seen that, even for a small number of steps $m \ll P$ and a single random vector, Lanczos impressively captures not only the support of the eigenvalue spectral density but also its shape. We note as discussed in Section 4.4.4, that the 30 Ritz values here do not span the top 30 eigenvalues even approximately.

Marchenko-Pastur: An equally important law for the limiting spectral density of many classes of matrices (constrained to be positive-definite, such as covariance matrices), is the Marchenko-Pastur law [Marchenko and Pastur, 1967]. Formally, given a matrix $\boldsymbol{X} \in \mathbb{R}^{P\times T}$ with i.i.d. zero mean entires with variance $\sigma^2 < \infty$. Let $\lambda_1 \geq \lambda_2, ... \geq \lambda_P$ be eigenvalues of $\boldsymbol{Y}_n = \frac{1}{T}\boldsymbol{X}\boldsymbol{X}^T$. The random measure $\mu_P(A) = \frac{1}{P}\#\{\lambda_j \in A\}$, $A \in \mathbb{R}$

Theorem 10. *Assume that $P, T \to \infty$ and the ratio $P/T \to q \in (0, \infty)$ (this*



(a) Stem $P = 225$ (b) Hist $P = 225$ (c) Stem $P = 10^4$ (d) Hist $P = 10^4$

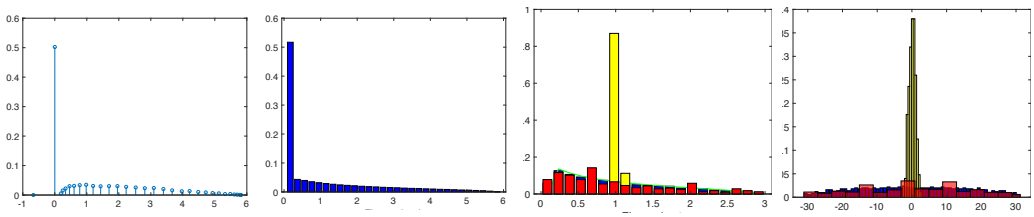
Figure 4.3: Lanczos stem plot for a single random vector with $m = 30$ steps compared to actual eigenvalue histogram for matrices of the form $\boldsymbol{H} \in \mathbb{R}^{P\times P}$. Each element is a drawn from a normal distribution with unit variance, which converges to the Wigner semi circle. Y-axis is the spectral density and the X axis is the eigenvalue. Note that whilst the histogram must sum to an area of 1, the discrete spectral density must sum to a total of 1. Hence we see that the essential information of the histogram is captured by the stem plot, especially in the high dimension limit.

is known as the Kolmogorov limit) then $\mu_P \to \mu$ *in distribution where*

$$\begin{cases} (1-\frac{1}{q})1_{0\in A} + \nu_{1/q}(A), & \text{if } q > 1 \\ \nu_q(A), & \text{if } 0 \le q \le 1 \end{cases} \tag{4.16}$$

$$d\nu_q = \frac{\sqrt{(\lambda_+ - x)(x-\lambda_-)}}{\lambda x 2\pi\sigma^2}, \lambda_\pm = \sigma^2(1 \pm \sqrt{q})^2 \tag{4.17}$$

Here, we construct a random matrix $\boldsymbol{X} \in \mathbb{R}^{P\times T}$ with independently drawn elements from the distribution $\mathcal{N}(0,1)$ and then form the matrix $\frac{1}{T}\boldsymbol{X}\boldsymbol{X}^T$, which is known to converge to the Marchenko-Pastur distribution. We use $P = \{225, 10000\}$ and $T = 2P$ and plot the associated histograms from full eigendecomposition in figures 4.5b & 4.5d along with their $m = 30, d = 1$ Lanczos stem counterparts in figures 4.5a & 4.5c. Similarly we see a faithful capturing not just of the support, but also of the general shape. We note that both for Figure 4.3 and Figure 4.5, the smoothness of the discrete spectral density for a single random vector increases significantly, even relative to the



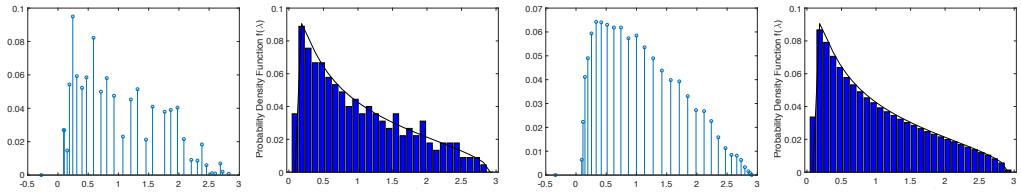
(a) Stem $P = 10^4$ (b) Hist,$P = 10^4$ (c) MP (d) Semi Circle

Figure 4.4: (a), (b) Lanczos stem plot for a single random vector with $m = 30$ steps compared to actual eigenvalue histogram for matrices of the form $\boldsymbol{H} \in \mathbb{R}^{P \times P}$, where $\boldsymbol{H} = \boldsymbol{X}\boldsymbol{X}^T/k$, where each element of $\boldsymbol{X}^{P \times k}$, $k = 0.5P$ is a drawn from a normal distribution with unit variance, converging to the Marchenko-Pastur distribution with $q = 0.5$. Y-axis is the spectral density and the X axis is the eigenvalue. Note that whilst the histogram must sum to an area of 1, the discrete spectral density must sum to a total of 1. (c),(d) Two randomly generated matrices $\boldsymbol{H} \in \mathbb{R}^{500 \times 500}$ with the histogram of the true eigenvalues in blue, the Lanczos estimate $m = 30, d = 1$ in red and the diagonal approximation in yellow.

histogram. We also run the same experiment for $P = 10000$ but this time with $T = 0.5P$ so that exactly half of the eigenvalues will be 0. We compare the Histogram of the eigenvalues in Figure 4.4b against its $m = 30, d = 1$ Lanczos stem plot in Figure 4.4a and find both the density at the origin, along with the bulk and support to be faithfully captured.

4.5.1 Comparison to Diagonal Approximations

As a proxy for deep neural network spectra, often the diagonal of the matrix [Bishop, 2006] or the diagonal of a surrogate matrix, such as the Fisher information, or that implied by the values of the Adam optimiser [Chaudhari et al., 2016] is used. We plot the true eigenvalue estimates for random matrices pertaining to both the Marchenko-Pastur 4.4c and the Wigner density 4.4d in blue, along with the Lanczos estimate in red and the diagonal approximation in yellow. We see here that the diagonal approximation in both



(a) Stem $P = 225$ (b) Hist,$P = 225$ (c) Stem $P = 10^4$ (d) Hist, $P = 10^4$

Figure 4.5: Lanczos stem plot for a single random vector with $m = 30$ steps compared to actual eigenvalue histogram for matrices of the form $\boldsymbol{H} \in \mathbb{R}^{P \times P}$, where the spectral density converges to the Marchenko-Pastur.

cases, fails to adequately the support or accurately model the spectral density, whereas the lanczos estimate is nearly indistinguishable from the true binned eigen-spectrum. This is of course obvious from the mathematics of the un-normalised Wigner matrix. The diagonal elements are simply draws from the normal distribution $\mathcal{N}(0, 1)$ and so we expect the diagonal histogram plot to approximately follow this distribution (with variance 1). However the second moment of the Wigner Matrix can be given by the Frobenius norm identity

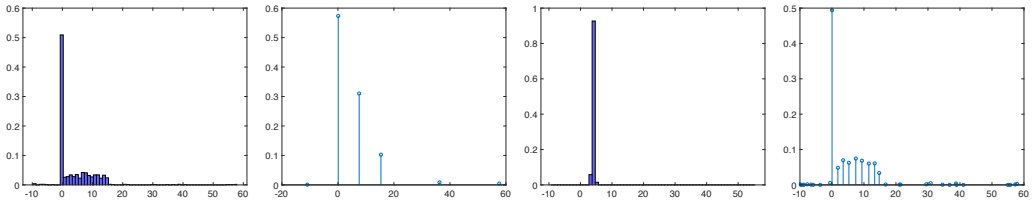
$$\mathbb{E}\left(\frac{1}{P}\sum_{i}^{P}\lambda_i^2\right) = \mathbb{E}\left(\frac{1}{P}\sum_{i,j=1}^{P}\boldsymbol{H}_{i.j}^2\right) = \mathbb{E}\left(\frac{1}{P}\chi_{P^2}^2\right) = P \tag{4.18}$$

Similarly for the Marchenko-Pastur distribution, We can easily see that each element of $\boldsymbol{H}$ follows a chi-square distribution of $1/T\chi_T^2$, with mean 1 and variance $2/T$.

4.5.2 Synthetic Example

The eigenspectrum of neural network Hessians often features a large spike at zero, a right-skewed bulk and some outliers [Sagun et al., 2016, 2018] [5] In order to simulate the spectrum of a neural network, we generate a Ma-

[5] Some examples of it can be found in later Sections on real-life neural network experiments - see Figure 4.8 for example.



(a) Hist of $\boldsymbol{H}$ (b) $m = 5$ Stem (c) Diagonal of $\boldsymbol{H}$ (d) $m = 30$ Stem

Figure 4.6: Generated matrices $\boldsymbol{H} \in \mathbb{R}^{1000\times1000}$ with known eigenspectrum and Lanczos stem plots for different values of $m = \{5, 15, 30\}$.

trix $\boldsymbol{H} \in \mathbb{R}^{1000\times1000}$ with 470 eigenvalues drawn from the uniform distribution from $[0, 15]$, 20 drawn from the uniform $[0, 60]$ and 10 drawn from the uniform $[-10, 0]$. The matrix is rotated through a rotation matrix $\boldsymbol{U}$, i.e. $\boldsymbol{H} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$ where $\boldsymbol{D}$ is the diagonal matrix consisting of the eigenvalues and the columns are Gaussian random vectors which are orthogonalised using Gram-Schmidt orthogonalisation. The resulting eigenspectrum is given in a histogram in Figure 4.6a and then using the same random vector, successive Lanczos stem plots for different number of iterations $m = [5, 30]$ are shown in Figure 4.6. Figure 4.6b, for a low number of steps, the degeneracy at $\lambda = 0$ is learned, as are the largest and smallest eigenvalues, some information is retained about the bulk density, but some of the outlier eigenvalues around $\lambda \approx 20$ and $\lambda \approx 30$ are completely missed out, along with all the negative outliers except the largest. For $m = 30$ even the shape of the bulk is accurately represented, as shown in Figure 4.6d. Here, we would like to emphasise that learning the outliers is important in the neural network context, as they relate to important properties of the network and the optimisation process [Ghorbani et al., 2019]. On the other hand, we note that the diagonal estimate in Figure 4.6c gives absolutely no spectral information, with no outliers shown (maximal and minimal diagonal elements being 5.3 and 3.3 respectively and it also gets the spectral mass at 0 wrong. This builds on Section 4.5.1, as

furthering the case against making diagonal approximations in general. In neural networks, the diagonal approximation is similar to positing no correlations between the weights. This is a very harsh assumption and usually a more reasonable assumption is to posit that the correlations between weights in the same layer are larger than between different layers, leading to a block diagonal approximation [Martens, 2016], however often when the layers have millions of parameters, full diagonal approximations are still used. [Bishop, 2006, Chaudhari et al., 2016].

4.6 Neural Network Examples

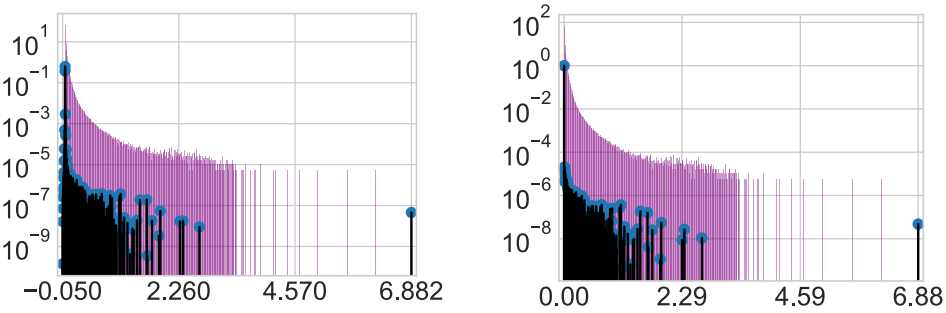
We showcase our spectral learning algorithm and visualisation tool on real networks trained on real data-sets and we test on VGG networks [Simonyan and Zisserman, 2015]. We train our neural networks using Stochastic Gradient Descent with momentum $\rho = 0.9$, using a linearly decaying learning rate schedule. We compare our method against recently developed open-source tools which calculate on the fly diagonal Hessian and Generalised Gauss-Newton diagonal approximations [Dangel et al., 2019].

4.6.1 VGG-16 CIFAR-100 Dataset

We train a 16-layer VGG network, comprising of $P = 15,291,300$ parameters on the CIFAR-100 dataset, using $\alpha_0 = 0.05$. Even for this relatively small model, the open-source Hessian and GGN exact diagonal computations require over 125GB of GPU memory and so to avoid re-implementing the library to support multiple GPUs and node communication we use the Monte Carlo approximation to the GGN diagonal against both our GGN-Lanczos and Hessian-Lanczos spectral visualisations. We plot a histogram of the

Monte Carlo approximation of the diagonal GGN (Diag-GGN) against both the Lanczos GGN (Lanc-GGN) and Lanczos Hessian (Lanc-Hess) in Figure 4.7. Note that as the Lanc-GGN and Lanc-Hess are displayed as stem plots (with the discrete spectral density summing to 1 as opposed to the histogram area summing to 1).

We note that the Gauss-Newton approximation quite closely resembles its



(a) Diag-GGN & Lanc-GGN

(b) Diag-GGN & Lanc-Hess

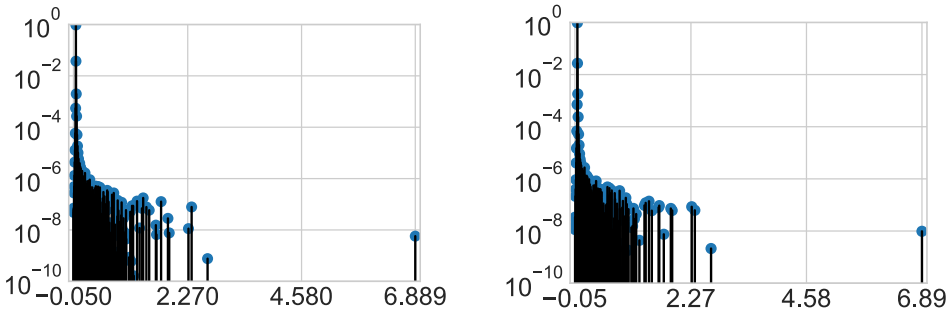
Figure 4.7: Diagonal Generalised Gauss-Newton monte carlo approximation (Diag-GGN) against $m = 100$ Lanczos using Gauss-Newton vector products (Lanc-GGN) or Hessian vector products (Lanc-Hess)

Hessian counterpart, capturing the majority of the bulk and the outlier eigenvectors at $\lambda_1 \approx 6.88$ and the triad near $\lambda_i \approx 2.29$. The Hessian does still have significant spectral mass on the negative axis, around 37%. However most of this is captured by a Ritz value at -0.0003, with this removed, the negative spectral mass is only 0.05%. However the Diag-GGN gives a very poor spectral approximation. It vastly overestimates the bulk region, which extends well beyond $\lambda \approx 1$ implied by Lanczos and adds many spurious outliers between 3 and the misses the largest outlier of 6.88. *Computational Cost* Using a single NVIDIA GeForce GTX 1080 Ti GPU, the Gauss-Newton takes an average 26.5 seconds for each Lanczos iteration with the memory usage 2850Mb. Using the Hessian takes an average of 27.9 seconds for each

Lanczos iteration with 2450Mb memory usage.

4.6.2 Effect of Varying Random Vectors

Given that the proofs for the moments of Lanczos matching those of the underlying spectral density, are true over the expectation over the set of random vectors and in practice we only use a Monte Carlo average of random vectors, or in our experiments using stem plots, just a single random vector. We have already given a proof dependent on the condition number in Section 2.5. Weiße et al. [2006], also arrive at the result that increasing the dimension should decrease the number of random vectors required, under the assumption that the $\text{Tr}(\boldsymbol{H}^2) \propto P$. We verify this high-dimensional result experimentally, by running the same spectral visualisation as in Section 4.6.1 but using two different random vectors. We plot the results in Figure 4.8. We find both figures 4.8a & 4.8b to be close to visually indistinguishable. There are minimal differences in the extremal eigenvalues, with former giving $\{\lambda_1, \lambda_n\} = \{6.8885, -0.0455\}$ and the latter $\{6.8891, -0.0456\}$, but the degeneracy at 0, bulk, triplet of outliers at 2.27 and the large outlier at 6.89 is unchanged.



(a) Vector 2 (b) Vector 3

Figure 4.8: VGG16 Epoch 300 end of training Lanczos stem plot for different random vectors.

5 | The effect of minibatching on curvature

In practical deep learning, we often sub-sample the dataset at every iteration. The extent of sub-sampling is often significant. For example the full ImageNet dataset is over one million images and a randomly selected subset at each gradient iteration may be in the hundreds or at most thousands. In this Chapter we consider how sub-sampling affects the loss surface and the implications for optimisation and the learning rate scaling rules for adaptive and non adaptive optimisers as a function of batch size.

Current deep learning papers using random matrix theory to analyse the loss surface typically assume that the weights and data are i.i.d. Gaussian, and that further under the assumption that the loss is independent of the data that the elements of the Hessian are i.i.d. Gaussian and that hence can be modelled as a random matrix. Not only is it intuitively jarring that a neural network that has learned important information should be structurally identical to a completely randomly initialised network trained on noise, but such assumptions fail to capture important components of the Hessian spectrum (such as well seperated outliers). Hence in this chapter we extend this formalism and consider the Hessian computed on the minibatch to be a random matrix under minibatch sampling. We consider this fluctuations matrix (the term that varies mini-batch to mini-batch) to be a random matrix (which is allowed to have some dependence structure and does not require identical elements), which perturbs the "signal" from the full dataset Hessian. Interestingly under certain assumptions, we can derive an analytical form for this perturbation.

It is worth mentioning that the for the reader well versed in random matrix

theory, that many of these results can be intuitively reasoned and understood. The originality in this Chapter, is solving the effect of mini-batching as the finite rank perturbation of a random matrix (which encapsulates the noise) and applying this to practical optimisation. We also derive random matrix theory assumptions under a different more known framework, that of optimisation.

5.1 Introduction

The observation that different neural network critical points post similar test set performance, has spawned an explosion of theoretical [Choromanska et al., 2015a,b, Pennington and Bahri, 2017] and empirical interest [Ghorbani et al., 2019, Li et al., 2017, Sagun et al., 2016, 2018, Wu et al., 2017], in their loss surfaces, typically through study of the eigenspectrum of the Hessian. Scalar metrics of the Hessian, such as the trace/spectral norm, have been related to generalisation [Keskar et al., 2016, Li et al., 2017]. Under a Bayesian [MacKay, 2003] and minimum description length framework [Hochreiter and Schmidhuber, 1997], flatter minima generalise better than sharp minima. Theoretical work on the Hessian of neural networks, has shown that all local minima are close to the global [Choromanska et al., 2015a] and that critical points of high index have high loss values [Pennington and Bahri, 2017]. second-order optimisation methods [Bottou et al., 2018], use the Hessian (or semi-positive-definite approximations thereof, such as the Fisher information matrix). They more efficiently navigate along narrow and sharp valleys, making significantly more progress per iteration [Martens, 2010, Martens and Sutskever, 2012, Martens and Grosse, 2015, Dauphin et al., 2014] than first order methods.

A crucial part of practical deep learning is the concept of sub-sampling or mini-batching. Instead of using the entire dataset of size N to evaluate the loss, gradient or Hessian at each training iteration, only a small randomly chosen subset of size $B \ll N$ is used. This allows faster progress and lessens the computational burden tremendously. However, despite its widespread use in optimisation, the precise characterisation of the effect of mini-batching on the loss landscape and implications thereof, have not been thoroughly investigated. In this chapter we show that,

- Under assumptions consistent with the optimisation paradigm, the Hessian computed on the minibatch is a random matrix under minibatch sampling.
- When well separated from the fluctuations matrix, which we define in Section 5.2.1 and which defines the perturbation between the minibatch matrix and the full dataset matrix ,the extremal eigenvalues of the batch Hessian are given by the extremal eigenvalues of the full Hessian plus a term proportional to the ratio of the *Hessian variance* to the batch size. We verify this empirically for the VGG-16 [Simonyan and Zisserman, 2015] on the CIFAR-100 dataset.
- This result predicts initial perfect scaling, diminishing returns and stagnation when increasing the batch size of Stochastic Gradient Descent training [Golmant et al., 2018, Shallue et al., 2018]. This result is crucial for understanding how to alter learning rate schedules when exploiting large batch training and data-parallelism, or when using limited GPU capacity for small or mobile devices.
- The minimum damping term of stochastic second-order methods [1], is

[1] often grid-searched as an extra hyper-parameter [Dauphin et al., 2014] or adjusted

inversely proportional to the batch size. For adaptive-gradient methods where the damping parameter is fixed to a small value, such as the Adam default settings, we derive and verify the efficacy of a square-root learning rate scaling rule with batch size.

Motivation: For samples drawn independently from the training set, the stochastic gradient $\boldsymbol{g}_i(\boldsymbol{w}) \in \mathbb{R}^{P\times 1}$ in expectation is equal to the empirical gradient $\mathbb{E}(\boldsymbol{g}_i(\boldsymbol{w})) = \boldsymbol{g}(\boldsymbol{w})$ [Boyd and Vandenberghe, 2009, Nesterov, 2013]. However for the sample inverse Hessian $\boldsymbol{H}_i^{-1}(\boldsymbol{w}) \in \mathbb{R}^{P\times P}$. $\mathbb{E}(\boldsymbol{H}_i^{-1}(\boldsymbol{w})) \neq \boldsymbol{H}^{-1}(\boldsymbol{w})$, as the inverse is not a linear operator. To notice this more clearly, imagine that we have an additive noise process hence $H_i(\boldsymbol{w}) = H(\boldsymbol{w}) + \epsilon(\boldsymbol{w})$, where the average eigenvalue of $\epsilon(\boldsymbol{w})$ is zero. Writing the eigenvalues of the perturbed matrix as $\lambda_i + \nu_i$, not that $\mathbb{E}(1/\lambda_i + \nu_i) \neq (1/\lambda_i)$ even if $\mathbb{E}\nu_i = 0$. By the spectral theorem, every Hermitian matrix, can be represented by its spectrum $\boldsymbol{H}(\boldsymbol{w}) = \sum_i^P \lambda_i \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T$ and hence the spectrum of $\boldsymbol{H}_i(\boldsymbol{w})$ differs from that of $\boldsymbol{H}(\boldsymbol{w})$. This follows because any Hermintian matrix can be written uniquely by its eigenvalue/eigenvector pairs and its inverse also. Mini-batching is prevalent in all [Martens and Grosse, 2015, Dauphin et al., 2014] deep learning second-order optimisation methods. Proofs of convergence for this class of methods explicitly require similarity between the sub-sampled and full dataset Hessian spectrum [Roosta-Khorasani and Mahoney, 2016]. Hence, understanding the spectral perturbations due to mini-batching is of great importance for second-order methods. For gradient methods on convex functions, the convergence rate, optimal and maximal learning rates are functions of the Lipschitz constant [Nesterov, 2013], which is the infimum of the eigenvalues of the Hessian in the weight manifold. Hence

during training [Martens, 2016]

understanding the largest eigenvalue perturbation due to mini-batching also has direct implications for their stability and convergence.

Related Work: To the best of our knowledge no prior work has theoretically or empirically compared the Hessian of the full dataset and that of a mini-batch and the consequences thereof. Hence the problem statement, theory and focus of this work are novel. Previous works focusing on the loss landscape structure as a function of loss value [Choromanska et al., 2015a, Pennington and Bahri, 2017], assume normality and independence of the inputs and weights [2]. Removing these assumptions is considered a major open problem [Choromanska et al., 2015b], addressed in the deep linear case with squared loss [Kawaguchi, 2016]. Furthermore, the spectra are not compatible with outliers, extensively observed in practice [Sagun et al., 2016, 2018, Ghorbani et al., 2019, Papyan, 2019]. We address both concerns, by considering a field dependence structure [Götze et al., 2012], non identical element variances and modeling the outliers explicitly as low-rank perturbations [Benaych-Georges and Nadakuditi, 2011]. This may be of more general use to the community outside of our applications. Our framework, prescribes a linear scaling rule up until a threshold for Stochastic Gradient Descent. [Krizhevsky, 2014, Goyal et al., 2017] also prescribe a linear scaling of learning rate with batch size, however it is justified under the unrealistic assumption that the gradient is the same at all points in weight space. [Jain et al., 2017] show linear parallelisation and then thresholding for least squares linear regression, assuming strong convexity. Our results hold for more general losses and does not assume strong convexity. Other

[2] and often even more assumptions, such as i.i.d. Hessian elements and and free addition [Pennington and Bahri, 2017] which means that we can simply add the spectra of two matrices

work which considers the effect of batch sizes on learning rate choices and various optimisation algorithms, considers a constant as opposed to evolving Hessian and relies on assumptions of co-diagonalizability of the Hessian and Covariance of the gradients [Zhang et al., 2019a], which is not necessary in our framework. We derive an inverse relationship between the damping coefficient of stochastic second-order methods and batch size, which is to our knowledge novel. For adaptive or stochastic second-order methods using small damping and small learning rates, our theory prescribes a square-root scaling procedure. [Hoffer et al., 2017] also prescribes a square-root scaling based on the co-variance of the gradients, for SGD but not adaptive methods. Jastrzkbski et al. [2017] investigate the effect of learning rate on curvature for the Hessian and note that the maximal spectral norm is quite close to the theoretical value of $\lambda_{max} = 1/\alpha$, however they neglect momentum in this calculation (which alters the formula [Nesterov, 2013]) and do not prescribe a learning rate schedule based on this observation.

5.2 Random matrix theoretic approach to the Batch Hessian

For an input, output pair $[\boldsymbol{x}, \boldsymbol{y}] \in [\mathbb{R}^{d_x}, \mathbb{R}^{d_y}]$ and a given prediction function $h(\cdot;\cdot) : \mathbb{R}^{d_x} \times \mathbb{R}^P \to \mathbb{R}^{d_y}$, we consider the family of prediction functions parameterised by a weight vector $\boldsymbol{w}$, i.e., $\mathcal{H} := \{h(\cdot;\boldsymbol{w}) : \boldsymbol{w} \in \mathbb{R}^P\}$ with a given loss function $\ell(h(\boldsymbol{x};\boldsymbol{w}),\boldsymbol{y}) : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$. In conjunction with statistical learning theory terminology, we denote the loss over our data generating distribution $\psi(\boldsymbol{x},\boldsymbol{y})$, as the *true risk*.

$$R_{true}(\boldsymbol{w}) = \int \ell(h(\boldsymbol{x};\boldsymbol{w}),\boldsymbol{y}) d\psi(\boldsymbol{x},\boldsymbol{y}), \tag{5.1}$$

with corresponding gradient $\boldsymbol{g}_{true}(\boldsymbol{w}) = \nabla R_{true}(\boldsymbol{w})$ and Hessian $\boldsymbol{H}_{true}(\boldsymbol{w}) = \nabla^2 R_{true}(\boldsymbol{w}) \in \mathbb{R}^{P\times P}$. Given a dataset of size N, we only have access to the *empirical risk*

$$R_{emp}(\boldsymbol{w}) = \sum_{i=1}^{N} \frac{1}{N} \ell(h(\boldsymbol{x}_i; \boldsymbol{w}), \boldsymbol{y}_i), \tag{5.2}$$

empirical gradient $\boldsymbol{g}_{emp}(\boldsymbol{w}) = \nabla R_{emp}(\boldsymbol{w})$ and empirical Hessian $\boldsymbol{H}_{emp}(\boldsymbol{w}) = \nabla^2 R_{emp}(\boldsymbol{w})$. To further reduce computation cost, often only the batch risk

$$R_{batch}(\boldsymbol{w}) = \frac{1}{B} \sum_{i=1}^{B} \ell(h(\boldsymbol{x}_i; \boldsymbol{w}), \boldsymbol{y}_i), \tag{5.3}$$

(where $B \ll N$) and the gradients $\boldsymbol{g}_{batch}(\boldsymbol{w})$, Hessians $\boldsymbol{H}_{batch}(\boldsymbol{w})$ thereof are accessed. The Hessian describes the curvature at that point in weight space $\boldsymbol{w}$ and hence the risk surface can be studied through the Hessian.

5.2.1 Properties of the fluctuation matrix

We write the stochastic batch Hessian as the deterministic empirical Hessian plus a perturbation due to the sampling noise.

$$\boldsymbol{H}_{batch}(\boldsymbol{w}) = \boldsymbol{H}_{emp}.(\boldsymbol{w}) + \boldsymbol{\epsilon}(\boldsymbol{w}) \tag{5.4}$$

Note that although we could write $\boldsymbol{H}_{emp}(\boldsymbol{w}) = \boldsymbol{H}_{batch}(\boldsymbol{w}) - \boldsymbol{\epsilon}(\boldsymbol{w})$, this treatment is not symmetric as $\boldsymbol{H}_{batch}(\boldsymbol{w})$ is dependent on $\boldsymbol{\epsilon}(\boldsymbol{w})$, whereas $\boldsymbol{H}_{emp}(\boldsymbol{w})$ is not. Rewriting the fluctuation matrix as $\boldsymbol{\epsilon}(\boldsymbol{w}) \equiv \boldsymbol{H}_{batch}(\boldsymbol{w}) - \boldsymbol{H}_{emp}(\boldsymbol{w})$ and assuming the mini-batch to be drawn independently from the dataset, we can infer

$$\boldsymbol{\epsilon}(\boldsymbol{w}) = \left(\frac{1}{B} - \frac{1}{N}\right) \sum_{j=1}^{B} \nabla^2 \ell(\boldsymbol{x}_j, \boldsymbol{w}; \boldsymbol{y}_j) - \frac{1}{N} \sum_{i=B+1}^{N} \nabla^2 \ell(\boldsymbol{x}_i, \boldsymbol{w}; \boldsymbol{y}_i)$$
$$\text{thus } \mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{j,k}) = 0 \text{ and } \mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{j,k})^2 = \left(\frac{1}{B} - \frac{1}{N}\right) \text{Var}[\nabla^2 \ell(\boldsymbol{x}, \boldsymbol{w}; \boldsymbol{y})_{j,k}]. \tag{5.5}$$

Where B is the batch size and N the total dataset size. The expectation is taken with respect to the data generating distribution $\psi(\boldsymbol{x}, \boldsymbol{y})$. In order for the variance in Equation 5.5 to exist, the elements of $\nabla^2 \ell(\boldsymbol{w}, \boldsymbol{w}; \boldsymbol{y})$ must obey sufficient moment conditions. This can either be assumed as a technical condition, or alternatively derived under the more familiar condition of L-Lipschitz continuity, as shown with the following Lemma.

Lemma 11. *For a Lipschitz-continuous empirical risk gradient and almost everywhere twice differentiable loss function $\ell(h(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y})$, the elements of the fluctuation matrix $\boldsymbol{\epsilon}(\boldsymbol{w})_{j,k}$ are strictly bounded in the range $-\sqrt{P}L \leq \boldsymbol{\epsilon}(\boldsymbol{w})_{j,k} \leq \sqrt{P}L$. Where P is the number of model parameters and L is a constant.*

Proof. As the gradient of the empirical risk is L Lipschitz continous, as the empirical risk a sum over the samples, the gradient of the batch risk is also Lipschitz continous. As the difference of two Lipschitz functions is also Lipschitz, by the fundamental theorem of calculus and the definition of Lipschitz continuity the largest eigenvalue λ_{max} of the fluctuation matrix $\boldsymbol{\epsilon}(\boldsymbol{w})$ must be smaller than L. Hence using the Frobenius norm we can upper bound the

matrix elements of $\boldsymbol{\epsilon}(\boldsymbol{w})$

$$\text{Tr}(\boldsymbol{\epsilon}(\boldsymbol{w})^2) = \sum_{j,k=1}^{P} \boldsymbol{\epsilon}(\boldsymbol{w})_{j,k}^2 = \boldsymbol{\epsilon}(\boldsymbol{w})_{j=j',k=k'}^2 + \sum_{j\neq j',k\neq k'}^{P} \boldsymbol{\epsilon}(\boldsymbol{w})_{j,k}^2 = \sum_{i=1}^{P} \lambda_i^2$$

$$\text{thus } \boldsymbol{\epsilon}(\boldsymbol{w})_{j=j',k=k'}^2 \leq \sum_{i=1}^{P} \lambda_i^2 \leq PL^2 \quad \text{and} \quad -\sqrt{P}L \leq \boldsymbol{\epsilon}(\boldsymbol{w})_{j=j',k=k'} \leq \sqrt{P}L. \tag{5.6}$$

□

As the domain of the Hessian elements under the data generating distribution is bounded, the moments of Equation 5.5 are bounded and hence the variance exists. We can even go a step further with the following extra lemma.

Lemma 12. *For independent samples drawn from the data generating distribution and an L-Lipschitz loss ℓ the difference between the empirical Hessian and Batch Hessian converges element-wise to a zero mean, normal random variable with variance $\propto \frac{1}{B} - \frac{1}{N}$ for large B, N.*

Proof. By Lemma 11, the Hessian elements are bounded, hence the moments are bounded and using independence of samples and the central limit theorem [Stein, 1972], $(\frac{1}{B} - \frac{1}{N})^{-1/2}[\nabla^2 R_{true}(\boldsymbol{w}) - \nabla^2 R_{emp}(\boldsymbol{w})]_{jk} \underset{a.s}{\longrightarrow} \mathcal{N}(0, \sigma_{jk}^2)$. □

5.2.2 The fluctuation matrix spectrum converges to the semi-circle law

To derive analytic results, we employ the Kolmogorov limit [Bun et al., 2017], where $P, B, N \to \infty$ but $P(\frac{1}{B} - \frac{1}{N}) = q > 0$. By Lemma 11, we have $\mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{j,k}) = 0$ and $\mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{j,k}^2) = \sigma_{j,k}^2$. To further account for dependence beyond the symmetry of the fluctuation matrix elements, we introduce the

σ-algebras

$$F^{(i,j)} := \sigma\{\boldsymbol{\epsilon}(\boldsymbol{w})_{kl} : 1 \leq k \leq l \leq P, (k,l) \neq (i,j)\}, \quad q \leq i \leq j \leq P \tag{5.7}$$

We can now state the following Theorem:

Theorem 13. *Under the conditions of Lemmas 11 and 12 along with the following technical conditions:*

(i) $\frac{1}{P^2}\sum_{i,j=1}^{P}\mathbb{E}|\mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}^2|F^{i,j}) - \sigma_{i,j}^2| \to 0,$

(ii) $\frac{1}{P}\sum_{i=1}^{P}|\frac{1}{P}\sum_{j=1}^{P}\sigma_{i,j}^2 - \sigma_\epsilon^2| \to 0$

(iii) $\max_{1\leq i\leq P}\frac{1}{P}\sum_{j=1}^{P}\sigma_{i,j}^2 \leq C$

when $P \to \infty$, the limiting spectra density $p(\lambda)$ of $\boldsymbol{\epsilon}(\boldsymbol{w}) \in \mathbb{R}^{P\times P}$ satisfies the semi circle law $p(\lambda) = \frac{\sqrt{4\sigma_\epsilon^2-\lambda^2}}{2\pi\sigma_\epsilon^2}$. Where $\mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}^2|F^{i,j})$ denotes the expectation conditioned on the sigma algebra, which is different to the unconditional expectation $\mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}^2|F^{i,j}) \neq \mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}^2) = \sigma_{i,j}^2$.

Proof. Lindenberg's ratio is defined as $L_P(\tau) := \frac{1}{P^2}\sum_{i,j=1}^{P}\mathbb{E}|\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}|^2 1(|\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}| \geq \tau\sqrt{P})$. By Lemma 12, the tails of the normal distribution decay sufficiently rapidly such that $L_P(\tau) \to 0$ for any $\tau > 0$ in the $P \to \infty$ limit. Alternatively, using the Frobenius identity and Lipschitz continuity $\sum_{i,j=1}^{P}\mathbb{E}|\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}|^2 1(|\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}| \geq \tau\sqrt{P}) \leq \sum_{i,j}^{P}\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}^2 = \sum_{i}^{P}\lambda_i^2 \leq PL^2$, $L_P(\tau) \to 0$ for any $\tau > 0$.

By Lemma 12 we also have $\mathbb{E}(\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}|F^{i,j}) = 0$. Hence along with conditions $i) - > iii)$ the matrix $\boldsymbol{\epsilon}(\boldsymbol{w})$ satisfies the conditions in Götze et al. [2012] and the and the limiting spectral density $p(\lambda)$ of $\boldsymbol{\epsilon}(\boldsymbol{w}) \in \mathbb{R}^{P\times P}$ converges to the semi circle law $p(\lambda) = \frac{\sqrt{4\sigma_\epsilon^2-\lambda^2}}{2\pi\sigma_\epsilon^2}$ [Götze et al., 2012].

Götze et al. [2012] use the condition $\frac{1}{P}\sum_{i=1}^{P}|\frac{1}{P}\sum_{j=1}^{P}\sigma_{i,j}^2 - 1| \rightarrow 0$, however this simply introduces a simple scaling factor, which is accounted for in condition *ii*) and the corresponding variance per element of the limiting semi-circle. □

We note that under the assumption of independence between all the elements of $\boldsymbol{\epsilon}(\boldsymbol{w})$ we would have obtained the same result, as long as conditions *ii*) and *iii*) we obeyed. So in simple words, condition *i*) merely states that the dependence between the elements cannot be too large. For example completely dependent elements have a second moment expectation that scales as P^2 and hence condition *i*) cannot be satistifed. Condition *ii*) merely states that there cannot be too much variation in the variances per element. and Condition *iii*) that the variances are bounded.

5.3 Main Result

Having shown that the limiting spectral density of the fluctuations matrix converges to the semi-circle, we are now in a position to present the main result of this paper.

Theorem 14. *Under the assumption that* $\boldsymbol{H}_{emp}$ *is of low-rank* $r \ll P$*, the extremal eigenvalues* $[\lambda'_1, \lambda'_P]$ *of the matrix sum* $\boldsymbol{H}_{batch}(\boldsymbol{w}) = \boldsymbol{H}_{emp}(\boldsymbol{w}) + \boldsymbol{\epsilon}(\boldsymbol{w})$*, where* $\lambda'_1 \geq \lambda'_2 ... \geq \lambda'_P$ *and* $\boldsymbol{\epsilon}(\boldsymbol{w})$ *is defined in Section 5.2.1 and obeys the conditions set out in Theorem 13, are given by*

$$\lambda'_1 = \left\{ \begin{array}{ll} \lambda_1 + \frac{P}{b}\frac{\sigma_\epsilon^2}{\lambda_1}, & \text{if } \lambda_1 > \sqrt{\frac{P}{b}}\sigma_\epsilon \\ 2\sqrt{\frac{P}{b}}\sigma_\epsilon, & \text{otherwise} \end{array} \right\}, \lambda'_P = \left\{ \begin{array}{ll} \lambda_P + \frac{P}{b}\frac{\sigma_\epsilon^2}{\lambda_P}, & \text{if } \lambda_P < -\sqrt{\frac{P}{b}}\sigma_\epsilon \\ -2\sqrt{\frac{P}{b}}\sigma_\epsilon, & \text{otherwise} \end{array} \right\}. \tag{5.8}$$

where $[\lambda_1, \lambda_P]$ *are the extremal eigenvalues of* $\boldsymbol{H}_{emp}(\boldsymbol{w})$*,* $b = B/(1 - B/N)$

and B is the batch-size.[3] *Recall that σ_ϵ is defined in Theorem 13, through the limiting spectral density $p(\lambda)$ of $\boldsymbol{\epsilon}(\boldsymbol{w})$.*

In order to prove Theorem 14 we utilise the following Lemma, which is taken from Benaych-Georges and Nadakuditi [2011] and for which we outline the proof in Appendix 5.4.1 for completeness.

Lemma 15. *Denote by $[\lambda'_1, \lambda'_P]$ the extremal eigenvalues of the matrix sum $\boldsymbol{M} = \boldsymbol{A} + \boldsymbol{\epsilon}(\boldsymbol{w})/\sqrt{P}$, where $\boldsymbol{A} \in \mathbb{R}^{P\times P}$ is a matrix of finite rank r with extremal eigenvalues $[\lambda_1, \lambda_P]$ and $\boldsymbol{\epsilon}(\boldsymbol{w}) \in \mathbb{R}^{P\times P}$ limiting spectral density $p(\lambda)$ satisfies the semi circle law $p(\lambda) = \frac{\sqrt{4\sigma_\epsilon^2 - \lambda^2}}{2\pi\sigma_\epsilon^2}$. Then we have*

$$\lambda'_1 = \left\{ \begin{array}{ll} \lambda_1 + \frac{\sigma_\epsilon^2}{\lambda_1}, & \text{if } \lambda_1 > 2\sigma_\epsilon \\ 2\sigma_\epsilon, & \text{otherwise} \end{array} \right\}, \; \lambda'_P = \left\{ \begin{array}{ll} \lambda_P + \frac{\sigma_\epsilon^2}{\lambda_P}, & \text{if } \lambda_P < -2\sigma_\epsilon \\ -2\sigma_\epsilon, & \text{otherwise} \end{array} \right\}. \tag{5.9}$$

We now proceed with the proof of Theorem 14:

Proof. The variance per element is a function of the batch size B and the size of the empirical dataset N, as given by Lemma 12. Furthermore, unravelling the dependence in P (which is simply the matrix dimension) due to the definition of the Wigner matrix (shown in Appendix 5.4) leads to Theorem 14. □

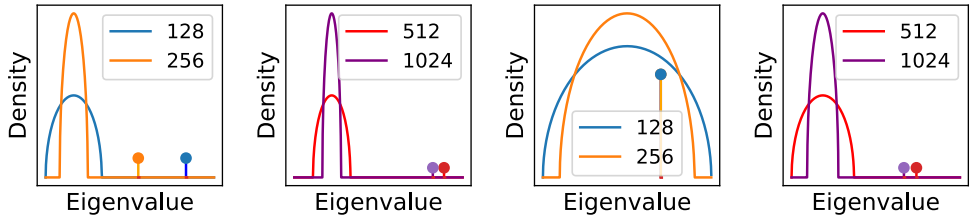
Comments on the Proof: Although for clarity we only focus on the extremal eigenvalues, the proof as shown in Section 5.4 holds for all outlier eigenvalues which are outside the spectrum of the fluctuation matrix. The assumption that either $\boldsymbol{H}_{emp}$ or $\boldsymbol{\epsilon}(\boldsymbol{w})$ are low-rank is necessary to use perturbation theory in the proof. This condition could be relaxed if a substantial

[3] Note that the factor $b = B/(1 - B/N)$ has appeared before in [Jastrzkbski et al., 2018, Jain et al., 2017].

part of the eigenspectrum of $\boldsymbol{H}_{emp}$ were considered to be mutually free with that of $\boldsymbol{\epsilon}(\boldsymbol{w})$ [Bun et al., 2017]. In Section 5.9 we derive a bound on the rank of a feed-forward network, which we show to be small for large networks and provide extensive experimental evidence that the full Hessian is in fact low-rank. In the special case that $\boldsymbol{\epsilon}(\boldsymbol{w})_{i,j}$ are i.i.d. Gaussian, the fluctuation matrix is the Gaussian Orthogonal Ensemble, proposed as the spectral density of the Hessian by Choromanska et al. [2015a]. In this case, Theorem 14 can be proved more succinctly, which we detail in full in the Appendix 5.4.

5.3.1 Illustration of the Key Result

We illustrate our key result (formalised in Theorem 14) in Figure 5.1.



(a) Linear (b) Threshold (c) square-root (d) Threshold

Figure 5.1: Variation of the spectral norm with batch size. The continuous region (bulk) corresponds to the fluctuation matrix induced by mini-batching (shown as a Semi Circle, whose width depends on the batch size) and the largest eigenvalue of the full Hessian is shown as a single peak. There are 3 major **scaling** regimes, which define how the largest eigenvalue decreases as the batch size increases.
(a) If the largest eigenvalue is well seperated from the spectrum of the fluctuation matrix, *the scaling is linear until a threshold* (b) beyond which there is no major change.
(c) If the largest eigenvalue is smaller than the largest bulk eigenvalue, the scaling is proportional to the square-root, until a threshold (d).
Crucially for deep learning optimisation, the scaling determines the allowed increase in learning rate

If the largest Hessian eigenvalue is well separated from the fluctuation matrix

(continuous spectral density), as shown in Figure 5.3a, then increasing the batch size, which reduces the spectral width of the fluctuation matrix, will have an approximately linear effect in reducing the spectral norm. This will hold up until a threshold, shown in Figure 5.1b, after which the spectral norm no longer appreciably changes in size. In the case that the spectral norm of the full dataset Hessian is smaller than that of the fluctuation matrix, shown in Figure 5.1c, the largest eigenvalue of the batch Hessian, which is given by the fluctuation matrix, will reduce as the square-root of the batch size, again up until a critical level shown in Figure 5.1d. As discussed in Section 5.5 the allowed increase in learning rate as a function of batch size is proportional to the scaling.

5.4 Proof of the Central Lemma

A full proof of Theorem 14, which rests heavily on disparate yet known results in the literature [Götze et al., 2012, Benaych-Georges and Nadakuditi, 2011, Bun et al., 2017] would span many dozens of pages, repeating prior work. We hence adopt an alternative proof strategy, which we hope is understandable and relatable to a machine learning audience, for which this work is intended. We first introduce a minimum amount of necessary random matrix theory background. We then prove Theorem 14, but under the stronger assumptions that the elements of the fluctuation matrix are i.i.d. Gaussian (the Gaussian Orthogonal Ensemble [Tao, 2012]). To understand why this makes sense, we consider the key ingredients of the proof

- The fluctuation matrix converges to the semi-circle law (which we introduce and explain in the next Section);
- the spectral perturbation low-rank empirical Hessian by the fluctuation

matrix can be computed analytically using perturbation theory;

- By Lemma 12, the scaling relationships which characterise the extent of the noise perturbation as a function of batch size can be analysed.

Hence the only difference between the simplified proof and Theorem 14, is that we have more general conditions for the convergence semi circle law, which are detailed extensively in Götze et al. [2012]. The other two key components proceed in an identical fashion.

5.4.1 Background

Following the notation of [Bun et al., 2017] the resolvent of a matrix H is defined as

$$\boldsymbol{G}_H(z) = (z\boldsymbol{I}_N - \boldsymbol{H})^{-1} \tag{5.10}$$

with $z = x + i\eta \in \mathbb{C}$. The normalised trace operator of the resolvent, in the $N \to \infty$ limit

$$\mathcal{S}_N(z) = \frac{1}{N}\text{Tr}[\boldsymbol{G}_H(z)] \xrightarrow{N\to\infty} \mathcal{S}(z) = \int \frac{\rho(\lambda)}{z-\lambda} du \tag{5.11}$$

is known as the Stieltjes transform of the eigenvalue density ρ. The functional inverse of the Siteltjes transform, is denoted the blue transform $\mathcal{B}(\mathcal{S}(z)) = z$. The $\mathcal{R}$ transform is thence defined as

$$\mathcal{R}(w) = \mathcal{B}(w) - \frac{1}{w} \tag{5.12}$$

Free random matrices: The property of freeness for non commutative random matrices can be considered analogously to the moment factorisation property of independent random variables. Let us denote the normalised

trace operator, which is equal to the first moment of the spectral density

$$\psi(H) = \frac{1}{N}\text{Tr}\boldsymbol{H} = \frac{1}{N}\sum_{i=1}^{N}\lambda_i = \int_{\lambda\in\mathcal{D}} d\mu(\lambda)\lambda \tag{5.13}$$

We say matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ for which $\psi(\boldsymbol{A}) = \psi(\boldsymbol{B}) = 0$ (We can always consider the transform $\boldsymbol{A} - \psi(\boldsymbol{A})\boldsymbol{I}$) are free if they satisfy for any integers $n_1..n_k$ with $k \in \mathbb{N}^+$

$$\psi(\boldsymbol{A}^{n_1}\boldsymbol{B}^{n_2}\boldsymbol{A}^{n_3}\boldsymbol{B}^{n_4}) = \psi(\boldsymbol{A}^{n_1})\psi(\boldsymbol{B}^{n_2})\psi(\boldsymbol{A}^{n_3})\psi(\boldsymbol{A}^{n_4}). \tag{5.14}$$

5.4.2 Proof of the Main Lemma

Recall that we wish to derive the values of the extremal eigenvalues of the matrix sum $\boldsymbol{M} = \boldsymbol{A} + \boldsymbol{\epsilon}(\boldsymbol{w})/\sqrt{P}$, where $\boldsymbol{\epsilon}(\boldsymbol{w})$ has a limiting spectral density given by the semi circle law. In this section we show by the definition of the Stieltjes transform (which has a one to one correspondence with the spectral density) that a finite rank perturbation of the Stieltjes transform (corresponding to the eigenvalues of the matrix $\boldsymbol{A}$) can be dealt with perturbation theory.

The Stijeles transform of the matrix $\boldsymbol{\epsilon}(\boldsymbol{w})$ with corresponding semi circle eigenvalue distribution can be written as [Tao, 2012]

$$\mathcal{S}_{\boldsymbol{\epsilon}}(z) = \frac{z \pm \sqrt{z^2 - 4\sigma_\epsilon^2}}{2\sigma_\epsilon^2}. \tag{5.15}$$

From the definition of the Blue transform, we hence have

$$z = \frac{\mathcal{B}_{\boldsymbol{\epsilon}}(z) \pm \sqrt{\mathcal{B}_{\boldsymbol{\epsilon}}^2(z) - 4\sigma_\epsilon^2}}{2\sigma_\epsilon^2} \therefore \;\; \mathcal{B}_{\boldsymbol{\epsilon}}(z) = \frac{1}{z} + \sigma_\epsilon^2 z \therefore \mathcal{R}_{\boldsymbol{\epsilon}}(z) = \sigma_\epsilon^2 z. \tag{5.16}$$

Computing the $\mathcal{R}$ transform of the rank 1 matrix $\boldsymbol{A}$ (which has a non-trivial eigenvalue $\lambda_1 > \sigma_\epsilon$) using the Stieltjes transform [Bun et al., 2017], we find the effect on the spectrum is given by:

$$\mathcal{S}_{\boldsymbol{A}}(u) = \frac{1}{N}\frac{1}{u-\lambda_1} + \left(1-\frac{1}{N}\right)\frac{1}{u} = \frac{1}{u}\left[1 + \frac{1}{N}\frac{\lambda_1}{1-u^{-1}\lambda_1}\right] \tag{5.17}$$

We can use perturbation theory similar to in Equation (5.16) to find the Blue and $\mathcal{R}$ transform which to leading order gives

$$\begin{aligned} \mathcal{B}_{\boldsymbol{A}}(\omega) &= \frac{1}{w} + \frac{\lambda_1}{N(1-\omega\lambda_1)} + \mathcal{O}(N^{-2}) \\ \mathcal{R}_{\boldsymbol{A}}(\omega) &= \frac{\lambda_1}{N(1-\omega\lambda_1)} + \mathcal{O}(N^{-2}) \end{aligned} \tag{5.18}$$

Setting $\omega = \mathcal{S}_{\boldsymbol{M}}(z)$ so

$$z = \mathcal{B}_{\boldsymbol{A}}(\mathcal{S}_{\boldsymbol{M}}(z)) + \frac{\lambda_1}{N(1-\lambda_1\mathcal{S}_{\boldsymbol{M}}(z))} + \mathcal{O}(N^{-2}) \tag{5.19}$$

using the ansatz of $\mathcal{S}_{\boldsymbol{M}}(z) = \mathcal{S}_0(z) + \frac{\mathcal{S}_1(z)}{N} + \mathcal{O}(N^{-2})$ we find that $\mathcal{S}_0(z) = \mathcal{S}_{\epsilon(w)}(z)$ and using that $\mathcal{B}'_{\boldsymbol{\epsilon}}(\mathcal{S}_{\boldsymbol{\epsilon}})(z)) = \frac{1}{\mathcal{S}_{\boldsymbol{\epsilon}}(z)}$, we conclude that

$$\mathcal{S}_1(z) = -\frac{\lambda_1 \mathcal{S}'_{\epsilon(w)}(z)}{1-\mathcal{S}_{\epsilon(w)}(z)\lambda_1} \tag{5.20}$$

and hence

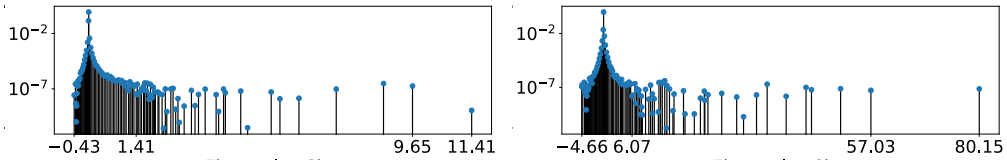
$$\mathcal{S}_{\boldsymbol{M}}(z) \approx \mathcal{S}_{\epsilon(w)}(z) - \frac{1}{N}\frac{\lambda_1 \mathcal{S}'_{\epsilon(w)}(z)}{1-\mathcal{S}_{\epsilon(w)}(z)\lambda_1} \tag{5.21}$$

In the large N limit the correction only survives if $\mathcal{S}_{\epsilon(w)}(z) = 1/\lambda_1$

$$\begin{aligned} \mathcal{S}_{\epsilon(w)}(z) = \frac{1}{\lambda_1} &\therefore \frac{2\sigma_\epsilon^2}{\lambda_1} = z \pm \sqrt{z^2 - 4\sigma_\epsilon^2} \\ \therefore z &= \lambda_1 + \frac{\sigma_\epsilon^2}{\lambda_1} \end{aligned} \tag{5.22}$$

The same proof also holds for $\lambda_P < -2\sigma_\epsilon$ and hence the second part of the Lemma also follows.

5.5 Experimental validation of Batch Hessians



(a) Hessian spectrum $N = 50,000$ (b) Hessian spectrum $N = 128$

Figure 5.2: Spectral Density of the Hessian at epoch 200, for different number of data-samples N on the VGG-16 network on the CIFAR-100 dataset

For simplicity, we do not analyse the added dependence between curvature and the samples due to batch normalisation [Ioffe and Szegedy, 2015] and hence adopt a reference model VGG-16 [Simonyan and Zisserman, 2015] on the CIFAR-100 dataset which does not utilise batch normalisation. We plot an example effect of the spectral broadening of the Hessian due to mini-batching, for a typical batch size of $B = 128$ in Figure 5.2. The magnitude of the extremal eigenvalues are significantly increased as are other outlier eigenvalues, such as the second largest. We estimate the mean of the continuous region (bulk) of the spectrum as where Ritz weight drops to below $1/P$. We see that the spectral width of this continuous region also increases.

5.6 Extension to Fisher information and other positive-definite matrices

In the case of Softmax regression, which is simply a 0 hidden layer neural network with cross-entropy loss, by the diagonal dominance theorem [Cover

and Thomas, 2012], the Hessian is semi-positive-definite and positive-definite with the use of $L2$ regularisation. Hence an underlying fluctuations matrix which contains negative eigenvalues is unsatisfactory and we extend our noise model to cover the positive semi definite case. Other common semi-positive-definite approximations to the Hessian in deep learning [Martens, 2014] include the Generalised Gauss-Newton (GGN) [Martens, 2010, Martens and Sutskever, 2012] and Fisher information [Martens and Grosse, 2015, Pennington and Worah, 2018]. For some common activations and loss functions typical in deep learning, such as the cross-entropy loss and sigmoid activation is equivalent to the Fisher information matrix [Pascanu and Bengio, 2014]. The Hessian may be expressed in terms of the activation σ at the output of the final layer $f(\boldsymbol{w})$ using the chain rule as

$$\boldsymbol{H}(\boldsymbol{w})_{ij} = \sum_{k=0}^{d_y}\sum_{l=0}^{d_y} \frac{\partial \sigma^2}{\partial f_l(\boldsymbol{w}) \partial f_k(\boldsymbol{w})}(f(\boldsymbol{w})) \frac{\partial f_l(\boldsymbol{w})}{\partial w_j}\frac{\partial f_k(\boldsymbol{w})}{\partial w_i} + \sum_{k=0}^{d_y} \frac{\partial \sigma}{\partial w_j}(f(\boldsymbol{w})) \frac{\partial f_k(\boldsymbol{w})^2}{\partial w_j \partial w_i} \tag{5.23}$$

The first term on the LHS of Equation 5.23 is known as the GGN matrix. The rank of a product is the minimum rank of its products so the GGN upper bounded by the $B \times d_y$. Following [Sagun et al., 2018] due to the convexity of the loss ℓ with respect with the output $f(\boldsymbol{w})$ we rewrite the GGN per sample as

$$\sum_{k,l=0}^{d_y} \sqrt{\frac{\partial \sigma^2}{\partial f_l(\boldsymbol{w}) \partial f_k(\boldsymbol{w})}(f(\boldsymbol{w}))} \frac{\partial f_l(\boldsymbol{w})}{\partial w_j} \times \sqrt{\frac{\partial \sigma^2}{\partial f_l(\boldsymbol{w}) \partial f_k(\boldsymbol{w})}(f(\boldsymbol{w}))} \frac{\partial f_k(\boldsymbol{w})}{\partial w_i} = \boldsymbol{J}_* \boldsymbol{J}_*^T \tag{5.24}$$

where $\boldsymbol{J}_*$ is the Jacobian transformed in order to retain a similarity for the GGN in the case of the squared loss function [Pennington and Bahri, 2017], under which it has the form $\boldsymbol{G}(\boldsymbol{w}) = \boldsymbol{J}\boldsymbol{J}^T$. There are many potential candidate noise models, such as the free multiplicative and information

plus noise [Bun et al., 2016, Hachem et al., 2013], typically for equations of the form in Equation 5.24, we would write $\boldsymbol{J}_{*batch} = \boldsymbol{J}_{*true}\boldsymbol{\epsilon}$ and hence $\boldsymbol{J}_{*batch}\boldsymbol{J}_{*batch}^T = \boldsymbol{J}_{*true}\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T\boldsymbol{J}_{*true}$ [Bun et al., 2017]. The corresponding analysis, gives a very similar result to Theorem 14. So the key take away is that *Independent of the exact limiting spectral density of the fluctuations matrix, we can consider the extremal eigenvalues of the True Hessian or generalised Gauss-Newton to be a low-rank perturbation of that fluctuations matrix.*

Since we are unable to find a reference that adequately derives or states the Stieltjes transform of the generalised non-unit variance Marchenko-Pastur density, we derive the result in full ourselves here. This derivation closely follows [Feier, 2012], but generalises the result. Note that Feier [2012] use a different convention for the Stieltjes transform

$$\mathcal{S}_P(z) = \int_{\mathbb{R}} \frac{1}{x-z} d\mu_P = \frac{1}{P}\operatorname{Tr}(\boldsymbol{M}_n/\sqrt{P} - zI)^{-1} \tag{5.25}$$

We consider a series of matrices

$$\boldsymbol{X}_N = \left(r_i^s/\sqrt{P}\right)_{1\leq i\leq P,\, 2\leq j\leq N} \tag{5.26}$$

where the entries r_i^s are 0 mean and variance σ^2 The Wishart matrix $\boldsymbol{W}_P = \boldsymbol{X}_P\boldsymbol{X}_P^T$ which has an (i,j) entry $\frac{1}{N}\sum_{s=1}^N r_i^s r_j^s$. Clearly, $\boldsymbol{W}_P$ can be written as the sum of rank-1 contributions $\boldsymbol{W}_P^s = (r_i^s r_j^s)_{1\leq i,j\leq P}$. Now as each element is of mean 0 and variance σ^2, the expectation of the sum of the elements squared is given by $P^2\sigma^4/T^2 = Tr([\boldsymbol{W}_P^s]^2) = \lambda^2$ and hence the one eigenvalue is given by $\lambda = \frac{P}{T}\sigma^2 = \beta\sigma^2$. For large P, by the weak law of large numbers, this is also true for a single realisation of $\boldsymbol{W}_P^s$. By the strong law of large numbers the column vectors $\boldsymbol{r}^s = [r_1^s ... r_P^s]^T$ and $\boldsymbol{r}^{s'}$ are almost surely orthogonal as

$P \to \infty$ and hence the matrices $\boldsymbol{W}_P^s$ are asymptotically free

The Stieltjes transform $\mathcal{S}(z)$ of $\boldsymbol{W}_P^s$

$$\frac{1}{P}\operatorname{Tr}(\boldsymbol{W}_P^s - z\boldsymbol{I})^{-1} = -\frac{1}{P}\sum_{k=0}^{\infty}\frac{\operatorname{Tr}(\boldsymbol{W}_P^s)^k}{z^{k+1}} = -\frac{1}{P}\left(\frac{P-1}{z} + \frac{1}{z-\beta\sigma^2}\right) \quad (5.27)$$

Solving the quadratic for z, completing the square, dropping low order terms in P and noting by the definition of the Stieltjes transform that for large $|z| \sim -\frac{1}{z}$

$$\begin{aligned} z &= \frac{P(s\beta\sigma^2 - 1) \pm \sqrt{P^2(s\beta\sigma^2-1)^2 + 4Ps(P-1)\beta\sigma^2}}{2Ps} \\ z &= \frac{P(s\beta\sigma^2 - 1) \pm \sqrt{P^2(s\beta\sigma^2+1)^2 - 4Ps\beta\sigma^2}}{2Ps} \\ z &\approx \frac{P(s\beta\sigma^2 - 1) - P(s\beta\sigma^2+1) - \frac{2Ps\beta\sigma^2}{s\beta\sigma^2+1}}{2Ps} = -\frac{1}{s} + \frac{\beta\sigma^2}{P(s\beta\sigma^2+1)} \end{aligned} \quad (5.28)$$

And hence as the $\boldsymbol{W}_P$ is the free convolution of the random matrices $\boldsymbol{W}_P^s$ we simply multiply the $\mathcal{R}$ transform of each matrix by N and as $\beta = P/N$

$$\begin{aligned} \mathcal{R}_{\boldsymbol{W}_P}(s) &= N \times \left(z - \frac{1}{s}\right) = \frac{N\beta\sigma^2}{P(s\beta\sigma^2+1)} = \frac{\sigma^2}{(s\beta\sigma^2+1)} \\ \mathcal{B}_{\boldsymbol{W}_P}(s) &= z = -\frac{1}{s} + \frac{\sigma^2}{(s\beta\sigma^2+1)} \end{aligned} \quad (5.29)$$

and hence

$$\mathcal{S}_{\boldsymbol{W}_P} = \frac{-(z+\sigma^2(1-\beta)) + \sqrt{(z+\sigma^2(1-\beta))^2 - 4\beta\sigma^2 z}}{2\beta\sigma^2 z} \quad (5.30)$$

From here, using the definition of the Stieltjes transform and the relationship to the spectral density, $\operatorname{Im}_{y\to 0}(\mathcal{S}_{\boldsymbol{W}_P}(x+iy))/2\pi i$

we have the celebrated generalised Marchenko-Pastur result.

$$\rho(y) = \frac{\sqrt{4\beta\sigma^2 y - (y + \sigma^2(1-\beta))^2}}{2\beta\sigma^2 y} \tag{5.31}$$

Noting that the Stieltjes transform from Feier [2012] is reversed in the convention of the sign [Bun et al., 2017], we take $z \to -z$. Now we apply for $\mathcal{T}(z) = z\mathcal{S}(z) - 1$ transform and the result from Benaych-Georges and Nadakuditi [2011], i.e. $\lambda_i' = \mathcal{T}(\frac{1}{\lambda_i})$, where the dash denotes the noisy transform.

$$\mathcal{T}(z) = \frac{z - \sigma^2(1+\beta) - \sqrt{(z + \sigma^2(1-\beta))^2 - 4\beta\sigma^2 z}}{2\beta\sigma^2} \tag{5.32}$$

Following through the algebra, with some simple cancellations, we arrive at a qualitatively similar result to Theorem 14.

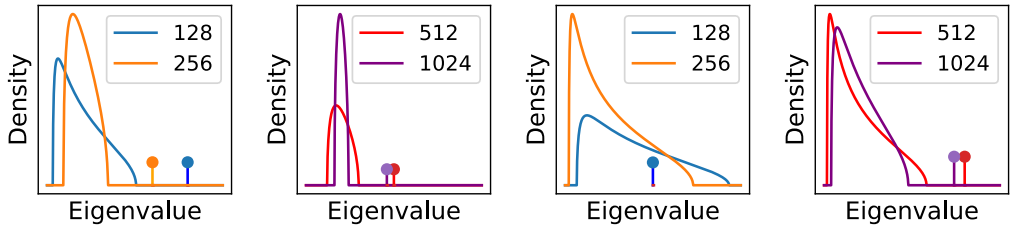
Theorem 16. *The extremal eigenvalue λ_1' of the matrix $\boldsymbol{G}_{batch}$, where $\boldsymbol{G}_{emp}$ has extremal eigenvalue λ_1, is given by*

$$\lambda_i' = \left\{ \begin{array}{ll} \frac{P\sigma^2}{b}(1 + \frac{\beta}{\lambda_i})(1 + \lambda_i), & \text{if } \lambda_i > \sigma\sqrt{\frac{P}{b}} \\ \sigma^2\frac{P}{b}, & \text{otherwise} \end{array} \right\}. \tag{5.33}$$

Proof Sketch. Combine the multiplicative perturbation results [Benaych-Georges and Nadakuditi, 2011], with the extension of the Marchenko-Pastur law for dependent entries [O'Rourke et al., 2012] □

We illustrate the result in Figure 5.3. Which is analogous to Figure 5.1, except for the Wigner semi circle is replaced with the Marchenko-Pastur density.

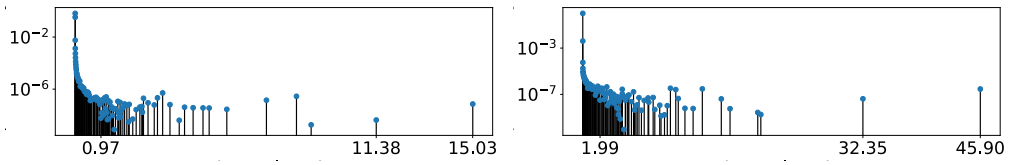
Remark. *We note that in the limit of $\sigma^2 \to 1$, ignoring P, b by folding them into σ, we have the same results as in [Benaych-Georges and Nadakuditi,*



(a) Linear scaling (b) Threshold (c) square-root (d) Threshold

Figure 5.3: Variation of the spectral norm with batch size. The continuous region corresponds to the fluctuation matrix induced by mini-batching (shown as the Marchenko-Pastur) and the largest eigenvalue of the full Hessian is shown as a single peak. Scaling refers to how the largest eigenvalue decreases as the batch size is increased.

2011], where they simply take $\theta = \theta - 1$. One point to note is that as the noise is multiplicative, a zero fluctuations matrix will ofcourse give all eigenvalues zero. To avoid this we could consider the matrix plus noise model were our matrix is now the Marchenko-Pastur.



(a) Empirical GGN $N = 50,000$ (b) Batch GGN $B = 128$

Figure 5.4: Spectral Density of the Generalised Gauss-Newton (GGN) at epoch 25 , on a VGG-16 on the CIFAR-100 dataset

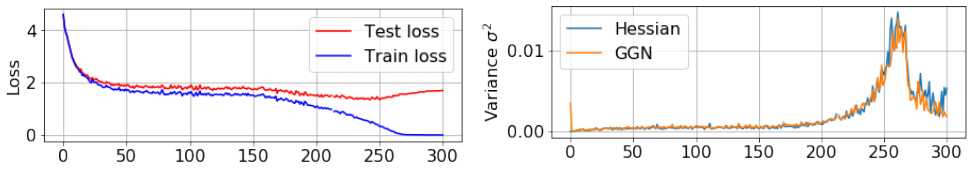
We plot an example of the generalised Gauss-Newton, which for cross-entropy loss and softmax activation is equal to the Fisher [Pascanu and Bengio, 2014] in Figure 5.4. We observe identical behaviour of bulk and outlier broadening.

We estimate the variance of the Hessian or GGN using stochastic trace estimation [Hutchinson, 1990b, Granziol and Roberts, 2017] in Algorithm 8, from which the variance per element can be inferred. We plot the evolution of the Hessian/GGN variance throughout an SGD training cycle in Figure 5.5.

This Figure implies that we expect the batch Hessian extremal eigenvalues to diverge from those of the empirical Hessian during training.

Algorithm 8 Calculate Hessian Variance

Input: Sample Hessian $\boldsymbol{H}_i \in \mathbb{R}^{P \times P}$
Output: Hessian Variance σ^2
$\boldsymbol{v} \in \mathbb{R}^{1 \times P} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$
$\boldsymbol{v} \leftarrow \boldsymbol{v}/||\boldsymbol{v}||$
Initialise $\sigma^2 = 0, i = 0$
for $i < N$ **do**
    $\sigma^2 \leftarrow \sigma^2 + \boldsymbol{v}^T \boldsymbol{H}_i^2 \boldsymbol{v}$
    $i \leftarrow i + 1$
end for
$\sigma^2 \leftarrow \sigma^2 - [\boldsymbol{v}^T (1/N \sum_{j=1}^{N} \boldsymbol{H}_j) \boldsymbol{v}]^2$

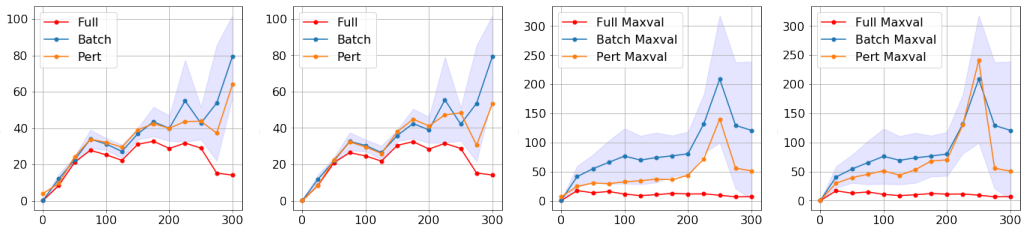


(a) Loss (b) Hessian Variance

Figure 5.5: Loss/variance evolution throughout SGD training VGG-16 CIFAR-100. X-axis denotes Epochs in Training

To test this hypothesis we run both the SGD and KFAC [Martens and Grosse, 2015] on the VGG-16 using CIFAR-100 and track the Hessian variance and full Hessian and the average of 10, $B = 128$ batch Hessian extremal eigenvalues and plot the results in Figure 5.6. The batch Hessian extremal eigenvalues have a large variance. This is to be expected as our results are in the limit $P, B \to \infty$ and corrections for finite B scale as $B^{-1/4}$ for matrices with finite 4th moments [Bai, 2008] which is $\approx 30\%$ for $B = 128$. Both the results from the additive noise process and multiplicative noise process give results within 1 standard deviation and follow the increase in variance of the Hessian in Figure 5.5. The multiplicative noise process gives a better

fit. Recent work shows the Hessian outliers to be attributable to the GGN component of the spectrum [Papyan, 2019]. Hence a positive semi definite noise process tailored for the GGN, would be expected to better estimate the outlier perturbations due to mini-batching, which we observe.



(a) KFAC $\lambda_1(\boldsymbol{H})$ (b) KFAC $\lambda_1(\boldsymbol{G})$ (c) SGD $\lambda_1(\boldsymbol{H})$ (d) SGD $\lambda_1(\boldsymbol{G})$

Figure 5.6: Evolution of the Variance σ and Maximal Eigenvalue λ_1 for both the Hessian $\mathbf{H}$ and GGN $\mathbf{G}$, during SGD and KFAC training on the VGG-16 using the CIFAR-100 dataset. Full, Batch and Pert refer to the full, batch and the theoretically predicted Hessian eigenvalues respectively.

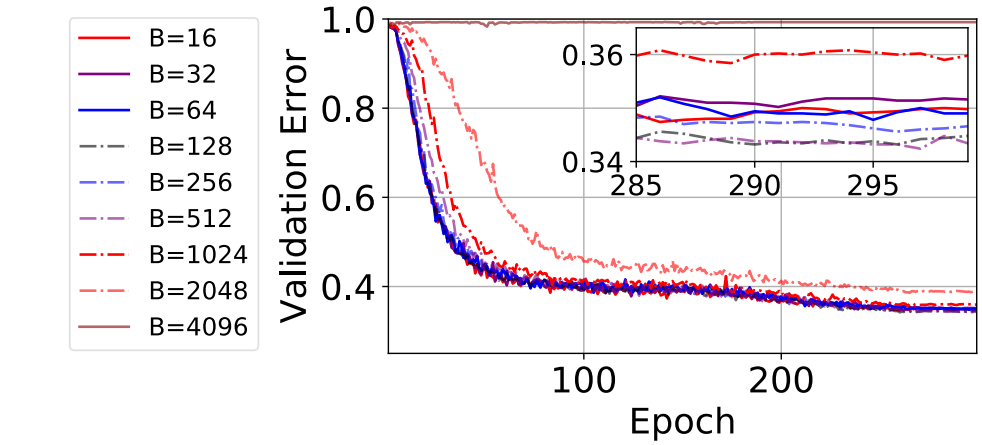
5.7 Application 1: SGD learning rates as a function of batch size

One key practical application of Section 5.2 for neural network training is its implications for learning rates as we alter the batch size. The change in loss to second order for a small step in the direction of the gradient is given by

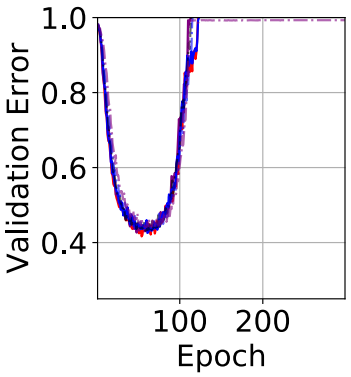
$$-\alpha||\boldsymbol{g}(\boldsymbol{w})||^2\left(1-\frac{\alpha\sum_i^P\lambda_i||\boldsymbol{\phi}_i\hat{\boldsymbol{g}}(\boldsymbol{w})||^2}{2}\right)\leq-\alpha||\boldsymbol{g}(\boldsymbol{w})||^2\left(1-\frac{\alpha\lambda_1}{2}\right) \quad (5.34)$$

and hence $\alpha < 2/\lambda_1$ to guarantee a decrease in loss , where the bound assumes that the entire gradient lies along the sharpest eigenvector and is hence very pessimistic. $\lambda_1(\boldsymbol{H}_{batch})$ and similarly all outlier eigenvalues of the batch Hessian are given by Theorem 14. A key term in Equation 5.34

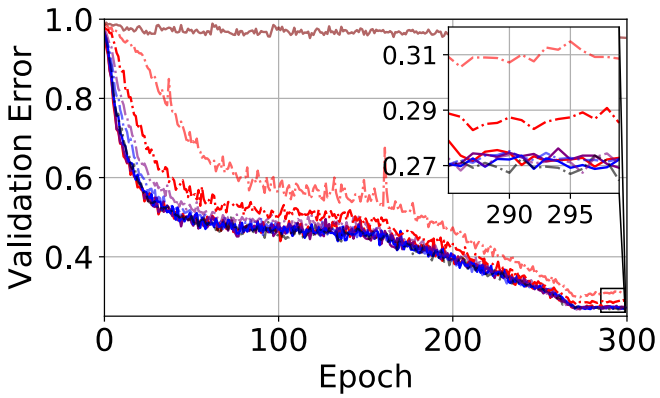
is the overlap between the eigenvectors $\boldsymbol{\phi}_i$ and the stochastic gradient, by which we mean the cosine angle between the unit vectors $\boldsymbol{\phi}_i$ and the unit gradient $\boldsymbol{g}(\hat{\boldsymbol{w}})$, shown to be large in practice [Ghorbani et al., 2019, Gur-Ari et al., 2018]. This indicates that the outlier broadening effect predicted by our framework (when there are well seperated outliers [4]), i.e. $\lambda_i* \approx \lambda_i + P\sigma^2/B\lambda_i$, is relevant to determining the maximal allowed learning rate. We observe spectral outliers (well separated from the bulk) in all our experiments, as shown in Figures 5.2 and 5.4, which is consistent with previous literature [Ghorbani et al., 2019, Papyan, 2019]. Large learning rates have been shown to induce implicit regularisation [Li et al., 2019]. In contrast, too small learning rates have been shown to lead to poor generalisation [Jastrzkbski et al., 2017, Berrada et al., 2018]. Hence learning the largest stable learning rate is an important practical question for neural network training. For small batch sizes the maximal learning rate is proportional to the batch size. This holds until the first term in Theorem 14 is no longer negligible with the latter. Whilst there are no strict guarantees for a decrease in loss (only a decrease to second order, which requires ignoring higher order terms), to guarantee a decrease we need to replace λ_1 with L, the co-efficient of Lipschitz continuity (which upper bounds $\lambda_1(\boldsymbol{w})\forall\boldsymbol{w}$). We note that for typical step size choices in practical schedules that we observe a monotonic decrease in loss (in the early stages) even for learning rates which are much larger than the largest eigenvalue observed throughout the trajectory and hence the approximation seems to hold well.



(a) NBN $\alpha_0 = \frac{0.05B}{128}$, $\gamma = 0$



(b) NBN $\alpha_0 = \frac{0.01B}{128}, \gamma = 0$



(c) BN $\alpha_0 = \frac{0.1B}{128}$, $\gamma = 5e^{-4}$

Figure 5.7: **Up to threshold, learning rates can be increased proportionally to the batch size increase without alterning performance (a)(c). Excessively large learning rates begin to diverge due to Hessian variance (b)** Validation error of the VGG-16, with (BN) and without (NBN) batch normalisation on the CIFAR-100 dataset. Corresponding weight decay γ and initial learning rate α_0

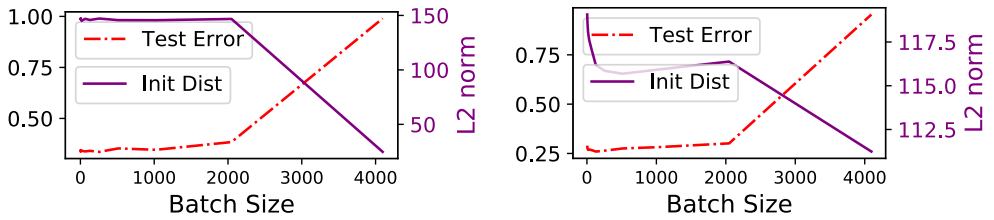
5.7.0.1 VGG-16, CIFAR-100

To validate this empirically, we train the VGG-16 on CIFAR-100, finding the maximal learning rate at which the network trains for $B = 128$. We then

[4]If there are no outliers, we expect the largest eigenvalue to decrease as the square-root of the batch size.

increase/decrease the batch size by factors of 2, proportionally scaling the learning rate. We plot the results in Figure 5.7a. The validation accuracy remains stable for all batch size values, until a small drop for $B = 1024$, a larger drop still for $B = 2048$ and for $B = 4096$ we see no training. Another theoretical prediction, validated by our experiment, is that if the Hessian variance increases during training (as observed in Figure 5.5), large learning rates which initially rapidly decrease the loss could become unstable later in training, which we observe in Figure 5.7b. To highlight the generality of this result, we include batch normalisation [Ioffe and Szegedy, 2015] and weight decay $\gamma = 0.0005$. In this case there is a greater range of permissible learning rates, so we grid search the best learning rate as defined by the validation error for $B = 128$ and use our derived linear scaling rule, with the results shown in Figure 5.7c, where we observe a similar pattern.

5.7.0.2 Alternative learning rate schedules and initialisation distance importance:



(a) VGG-16 (b) VGG-16BN

Figure 5.8: Test error as a function of initialisation distance for both the VGG-16 and VGG-16BN, for the CIFAR-100 dataset. Learning rate is scaled linearly with batch size.

One implicit assumption in Section 5.7 is that the largest learning rate which trains with inherent stability gives the best result. This informs our work as to how we should scale this rate as the batch size is increased. However

it makes sense to consider how alternative more conservative scaling rules might fare and whether they impact performance. In this section we also consider whether increased distance from Initialisation, as posited in Hoffer et al. [2017] is relevant for generalisation.

We do this for the VGG-16 on the CIFAR-100 dataset. Against a baseline validation accuracy of 65.82% for $B = 128$. For the $B = 1024$ case, our theoretically justified linearly increased learning rate of 0.08 gives an accuracy of 64.35%, whereas using the square-root rule [Hoffer et al., 2017] suggestion of 0.028 only gives 61.08%, we note from Figure 5.2 that there are many well separated outliers and hence that we expect Theorem 14 to hold and hence give linear scaling. On the held out test set, the linear scaling solution has an error of 34.64% and a distance of 145.76 in $L2$ norm from the initialisation, whereas the square-root scaled solution has an error of 37.55% and a distance of 67.44 from initialization. This indicates that as argued in Hoffer et al. [2017] that distance from initilisation seems to play an important role for generalisation. We test this further by looking at the initialisation distance across the set of similar test performing solutions for a constant learning rate to batch size ratio. Interestingly for the VGG-16 without batch normalisation as shown in Figure 5.8a there is a strong link between initialisation distance in $L2$ norm and the test error. This relationship is much weaker and much smaller in magnitude when batch normalisation is utilised, as shown in Figure 5.8b. We even see the distance from initialisation increasing as test error also increases.

5.7.0.3 WideResNet-28 × 10, CIFAR-100 and ImageNet-32

We repeat the experiment on the WideResNet-28 × 10 [Zagoruyko and Komodakis, 2016] on both the CIFAR-100 and ImageNet 32 × 32 [Chrabaszcz

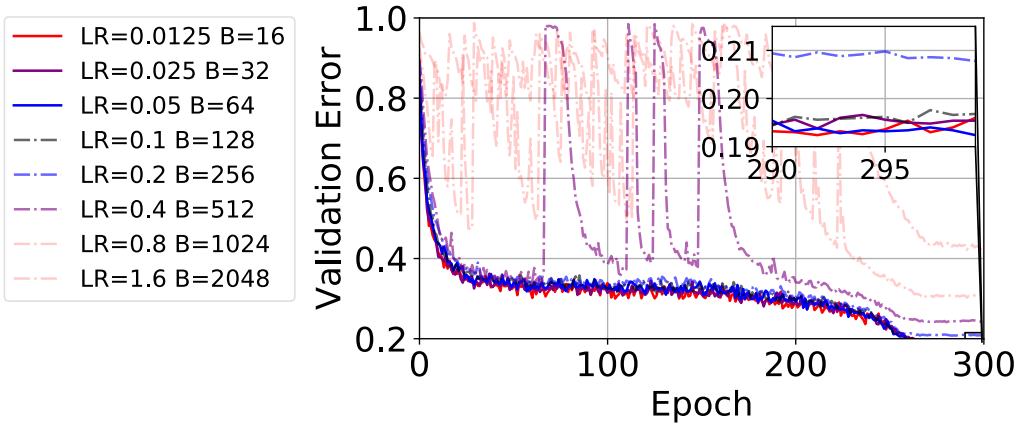


Figure 5.9: WideResNet-28 $\times$ 10 with batch normalisation, CIFAR-100 validation performance for various batch sizes, $\alpha = \frac{0.1B}{128}, \gamma = 5 \times 10^{-4}$

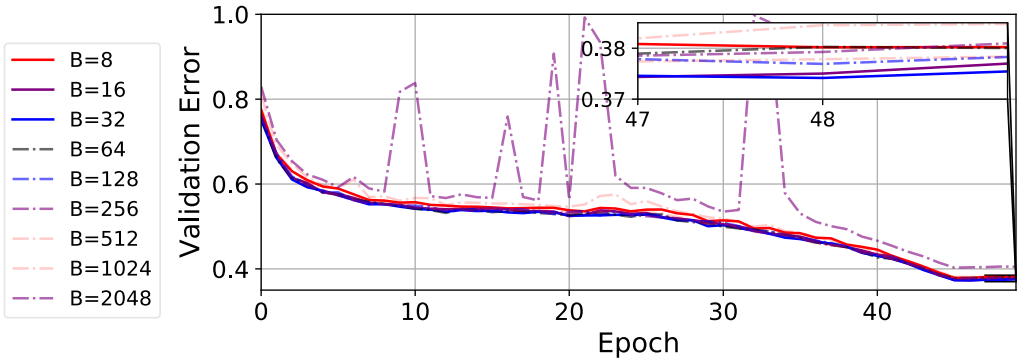


Figure 5.10: WideResNet-28 $\times$ 10 with batch normalisation, ImageNet-32 validation performance for various batch sizes, $\alpha = \frac{0.1B}{128}, \gamma = 5 \times 10^{-5}$

et al., 2017] datasets shown in Figures 5.9 and 5.10. Unlike the VGG without batch normalisation, where unstable trajectories diverge or with batch normalisation do not train. Highly unstable oscillatory WideResNet trajectories converge with learning rate reduction, however they never reach peak performance. The test performance is stable for a variety of learning rates with fixed learning rate to batch size ratio. Again confirming the validity of the linear scaling rate rule until a threshold.

5.8 Application 2: learning rates/damping for 2^{nd} order/adaptive optimisation

The framework derived in Section 5.2, also allows us to predict scaling relations between the damping constant, learning rate and batch size in stochastic second-order and adaptive methods. Damping is a key hyper-parameter in second order methods which limits the movement of the optimiser in any direction. Whilst largely seen as a heuristic for numerical stability, we argue that it is crucial due to the sampling noise in stochastic second order deep learning.

Damping: second-order optimisation methods minimise the quadratic around a small perturbation of the Loss $L(\boldsymbol{w}+\boldsymbol{\delta w})$

$$\boldsymbol{\delta w}^{*}=\arg\min_{\boldsymbol{\delta w}}\left(L(\boldsymbol{w})+\nabla L(\boldsymbol{w})\boldsymbol{\delta w}+\frac{1}{2}\boldsymbol{\delta w}^{T}\boldsymbol{H}(\boldsymbol{w})\boldsymbol{\delta w}\right)=-\boldsymbol{H}^{-1}(\boldsymbol{w})\nabla L(\boldsymbol{w}) \tag{5.35}$$

Where in practice $\boldsymbol{H}^{-1}=\sum_{i}^{P}(\lambda_i+\delta)^{-1}\phi_i\phi_i^{T}$ and δ (the damping coefficient) keeps the optimiser within a trust region [Martens and Sutskever, 2012], limiting its ability to take overly large steps in locally flat directions. It is typically set at constant values and grid searched [Dauphin et al., 2014, Vinyals and Povey, 2012]. Hence δ introduces partial adaptivity, at 0 we have a fully second-order method and at $\delta \gg \lambda_{max}$ we resort to SGD with learning rate α_0/δ. Using our additive noise model from Section 5.2, we can derive an analytic Equationfor the minimum damping required for a given batch size. The overlap between the batch extremal eigenvectors and their full dataset counterparts is given as $|\boldsymbol{\phi}_i^{T}\hat{\boldsymbol{\phi}}_i|^2=1-P\sigma^2/b\lambda_i^2$ [Benaych-Georges and Nadakuditi, 2011]. By considering the change in loss for a generic second-

order optimiser, writing $\boldsymbol{H}_{emp} = \sum_i \lambda_i \boldsymbol{\psi}_i^T \boldsymbol{\psi}$ and writing the noisy estimated eigenvalue/eigenvector from the optimiser as $\lambda_i, \boldsymbol{\phi}_i$, we have

$$L(\boldsymbol{w}_{k+1}) - L(\boldsymbol{w}) = \sum_i^P \frac{\alpha_0 |\boldsymbol{\phi}_i^T \nabla L(\boldsymbol{w})|^2}{\lambda_i + \delta} \left(1 - \frac{\alpha_0}{2(\lambda_i + \delta)} \sum_\mu \lambda_\mu |\boldsymbol{\psi}_u^T \boldsymbol{\phi}_i|^2)\right)$$
$$\therefore \delta > \frac{\alpha_0 P \sigma^2}{2b} \tag{5.36}$$

Where we assume δ to be sufficiently large. Hence the largest loss increase is due estimated flat directions having residual overlap with the sharpest direction of the loss. The total residual overlap is $P\sigma^2/b\lambda_i^2$ and for typical positive-definite approximations [Martens, 2016, Vinyals and Povey, 2012, Dauphin et al., 2014], in the worst case, $\lambda_i \ll \delta$ in these directions, from which the result follows. To verify the validity of this relation, we run both the KFAC [Martens and Grosse, 2015] and Adam [Kingma and Ba, 2014] optimisers on the VGG-16 with no weight decay, with $\alpha = 1$, grid searching δ until we train to good accuracy[5]. The the batch size is increased/decreased by factors of 2 and the damping in inverse proportion. We display the training/validation errors for in Figures 5.13/5.14 for KFAC and 5.11/5.12 for Adam respectively, where we note good agreement with theory.

Damping and Generalisation: For Adam $\delta = 10^{-8}$ is the default setting, which in practice is trained with a low learning rate to compensate. We show the training and testing curves of typical learning rate values used $[1e^{-4}, 3e^{-4}]$ in Figures 5.11 and 5.12. Although not faster or better initially, such curves soon train better and generalise worse. We do not employ weight decay in this experiment, hence regularisation implementations [Loshchilov and Hutter, 2019] are not relevant. each optimisation step

[5]In the adam optimiser δ is given as ϵ and usually set to a default value of 10^{-8}

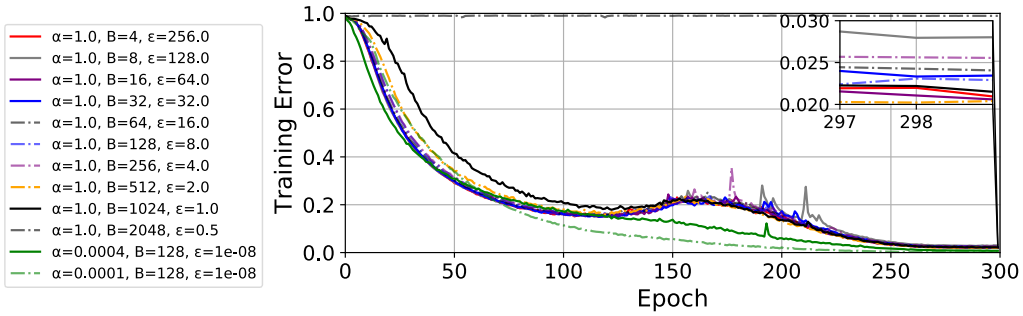


Figure 5.11: Training error of the Adam optimiser for various learning rates, scaled linearly, trained with large learning rate α and large damping constant ϵ on the VGG-16 without batch normalisation on the CIFAR-100 dataset

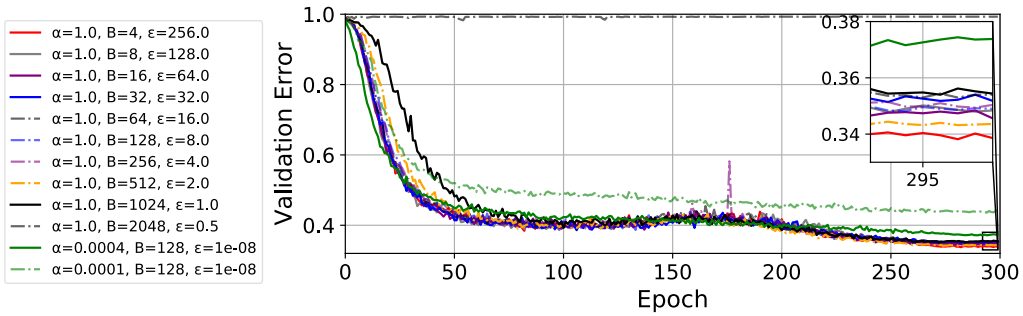


Figure 5.12: Validation error of the Adam optimiser for various damping rates, scaled linearly, trained with large learning rate $\alpha = 1$ and large damping constant ϵ on the VGG-16 without batch normalisation on the CIFAR-100 dataset

is $-\sum_i^P (\lambda_i/\alpha_0 + \delta/\alpha_0)^{-1}\phi_i\phi_i^T\nabla L(\boldsymbol{w})$ and hence for fixed δ/α_0, smaller α_0 takes smaller directions in the sharp loss directions, which seems to impact generalisation. Tuning the damping coefficient has been shown to improve generalisation in [Choi et al., 2019]. We develop a fully fledged theory of this phenomenon in Chapter 6.

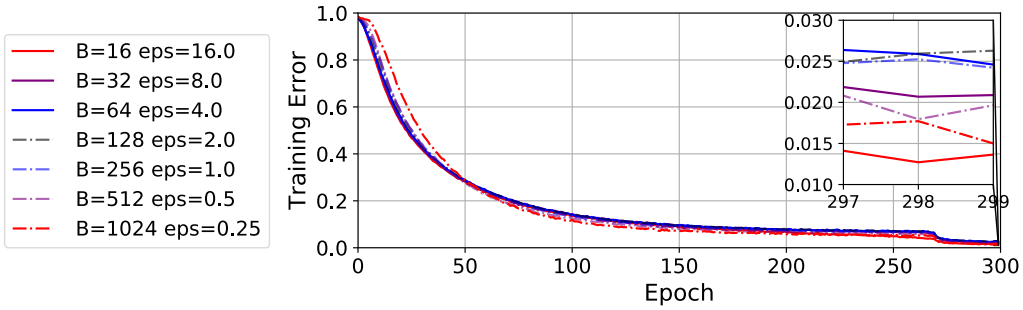


Figure 5.13: Training error of the KFAC optimiser for various damping rates, scaled linearly with batch size, trained with large learning rate $\alpha = 1$ and large damping constant ϵ on the VGG-16 without batch normalisation on the CIFAR-100 dataset

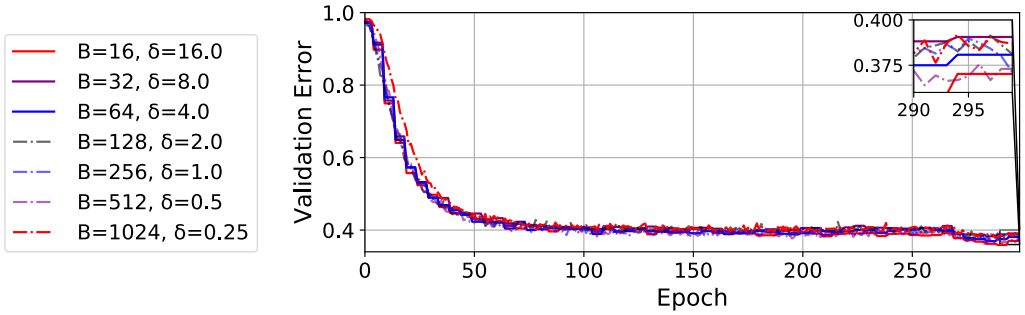


Figure 5.14: Validation error of the KFAC optimiser for various damping rates, scaled linearly with batch size, trained with large learning rate $\alpha = 1$ and large damping constant ϵ on the VGG-16 without batch normalisation on the CIFAR-100 dataset

5.8.1 Square-root learning rate scaling for adaptive optimisers with small damping

From Equation 5.36, for small δ the greatest loss changes could result from large steps along flat directions. From Theorem 14 we expect the sharpness of some of these directions to grow inversely proportionally to the square-root of the batch size. This implies that if our damping is small, we should scale the learning rate as the square-root of batch size. We try this for

the Adam optimiser with the default damping of 10^{-8}. We grid search the largest stable learning rate for a batch size of 128 (which is $\alpha = 0.0004$) and then increase/decrease the batch size by a factor of 2, scaling the learning rate proportional to the square-root of the batch size change. We show the results in Figure 5.15, which validates our hypothesis, whereas the linear prescription fails completely.

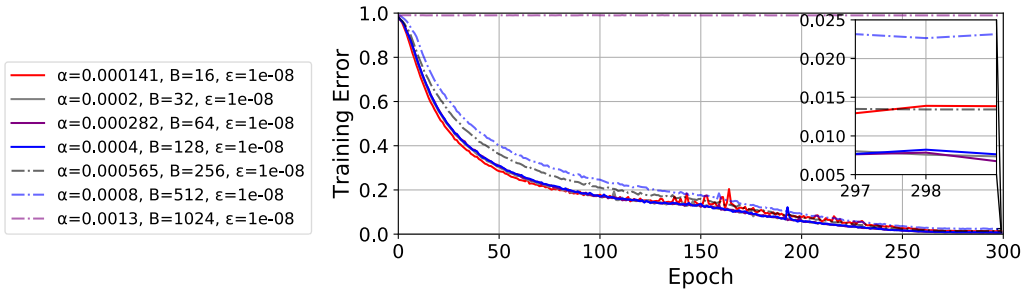


Figure 5.15: Training error of the Adam optimiser for various learning rates, scaled with the square-root of the batch size, trained with a small learning rate α and small damping constant $\epsilon = 10\times^{-8}$ on the VGG-16 without batch normalisation on the CIFAR-100 dataset

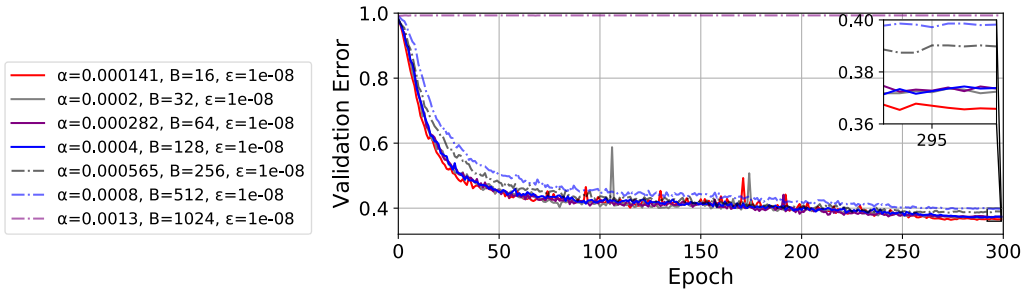


Figure 5.16: Validation error of the Adam optimiser for various learning rates, scaled with the square-root of the batch size, trained with a small learning rate α and small damping constant $\epsilon = 10\times^{-8}$ on the VGG-16 without batch normalisation on the CIFAR-100 dataset

5.8.2 Experimental Details

On the CIFAR-100 and ImageNet-32 datasets (45,000 training samples and 5,000 validation samples or $1m$ training samples and $50,000$ validation samples respectively) using SGD and K-FAC optimisers. For both SGD and K-FAC, we use the following learning rate schedule:

$$\alpha_t = \begin{cases} \alpha_0, & \text{if } \frac{t}{T} \leq 0.5 \\ \alpha_0[1 - \frac{(1-r)(\frac{t}{T}-0.5)}{0.4}] & \text{if } 0.5 < \frac{t}{T} \leq 0.9 \\ \alpha_0 r, & \text{otherwise} \end{cases} \tag{5.37}$$

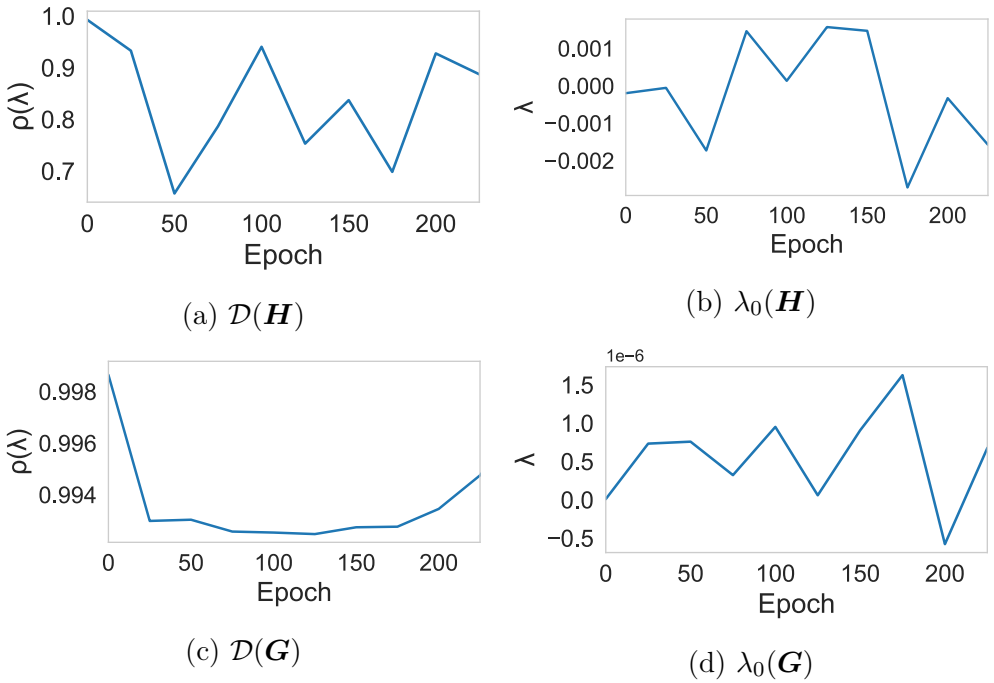
We use learning rate ratio $r = 0.01$ and total number of epochs budgeted $T = 300$. We further use momentum $\rho = 0.9$, a weight decay coefficient of 0.0005 and data-augmentation on PyTorch [Paszke et al., 2017]. We further set the inversion frequency to be once per 100 iterations for K-FAC.

5.9 Low-Rank Approximation

One of the key ingredients to proving 14, as shown in 5.2 is the use of perturbation theory. This requires either the fluctuations matrix or the full empirical Hessian to be low-rank. In our work, we consider the empirical Hessian to be low-rank. The rank degeneracy of small neural networks has already been discovered and discussed in Sagun et al. [2018] and reported for larger networks using spectral approximations in Ghorbani et al. [2019], Papyan [2019]. We provide extensive experimental validation for both the VGG-16 and PreResNet-110 on the CIFAR-100 datasets in Sections 5.9.1 and 5.9.2, along with a theoretical argument and rank bound for feed forward neural networks with cross-entropy loss in Section 5.9.4.

Experimental Setup: Given that Hessians have P^2 elements with a full inversion cost of $\mathcal{O}(P^3)$ which is infeasible for large neural networks. Counting the number of 0 eigenvalues (which sets the degeneracy) is not feasible in this manner. Furthermore, there would still be issues with numerical precision, so a threshold would be needed for accurate counting. Hence, based on our understanding of the Lanczos algorithm, discussed in Chapter 4, we propose an alternative method. We know that m steps of the Lanczos method, gives us an m-moment-matched spectral approximation of the moments of $\boldsymbol{v}^T\boldsymbol{H}\boldsymbol{v}$, where in expectation over the set of zero mean unit variance random vectors this is equal to the spectral density of $\boldsymbol{H}$. Each eigenvalue, eigenvector pair estimated by the Lanczos algorithm is called a Ritz-value/Ritz-vector. We hence take $m \gg 1$, where typically and for consistency we take $m = 100$ in our experiments. We then take the Ritz value closest to the origin and take that as a proxy for the 0 eigenvalue and report its weight. One weakness of this method is that for a large value of m, since the Lanczos algorithm finds a discrete moment-matched spectral algorithm, is that the spectral mass near the origin, may split into multiple components and counting the largest thereof or closest to the origin may not be sufficient. We note this problem both for the PreResNet-110 and VGG-16 on the CIFAR-100 dataset shown in Figure 5.17. Significant drops in degeneracy occur at various points in training and occur in tandem with significant changes in the absolute value of the Ritz value of minimal magnitude. This suggests the aforementioned splitting phenomenon is occurring. This issue is not present in the calculation of the generalised Gauss-Newton, as the spectrum is constrained to be positive-definite, so there is a limit to the extent of splitting that may occur. In order to remedy this problem, for the Hessian we calculate the combination of the two closest Ritz values around the centre

and combine their mass. We consider this mass and the weighted average of their values as the degenerate mass. An alternative approach could be to kernel smooth the Ritz weights at their values, but this would involve another arbitrary hyper-parameter σ.



(a) $\mathcal{D}(\boldsymbol{H})$

(b) $\lambda_0(\boldsymbol{H})$

(c) $\mathcal{D}(\boldsymbol{G})$

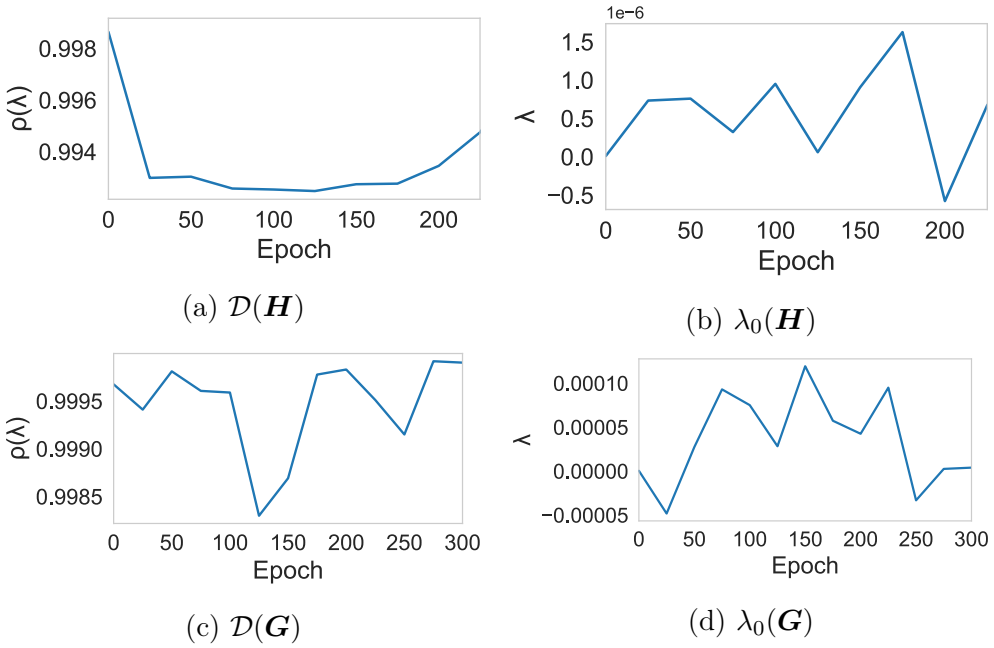
(d) $\lambda_0(\boldsymbol{G})$

Figure 5.17: Rank degeneracy $\mathcal{D}$ evolution of both the Hessian $\boldsymbol{H}$ and GGN $\boldsymbol{G}$ throughout training using the PreResNet-110 on the CIFAR-100 dataset, the weight $\rho(\lambda)$ corresponds to the spectral mass of the Ritz value $\mathcal{D}(\lambda_0)$ which is closest to the origin.

5.9.1 VGG16

For the VGG-16, which forms the reference model for this Chapter, we see that for both the generalised Gauss-Newton (shown in Figure 5.18a) and the Hessian (shown in Figure 5.18c) that the rank degeneracy is extremely high. For the GGN, the magnitude of the Ritz value which we take to be the origin, is extremely close to the threshold for GPU precision, as shown in Figure

5.18b. For the Hessian, for which we combine the two smallest absolute value Ritz values, we have as expected an even larger spectral degeneracy. The weighted average, also gives a value very close to 0, as shown in Figure 5.18d. Although the combined weighted average is much closer to the origin, than that of the lone spectral peak, shown in Figure 5.17, which indicates splitting, we do not get as close to the GPU precision threshold.



(a) $\mathcal{D}(\boldsymbol{H})$ (b) $\lambda_0(\boldsymbol{H})$

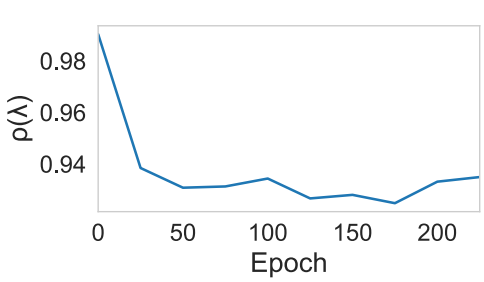
(c) $\mathcal{D}(\boldsymbol{G})$ (d) $\lambda_0(\boldsymbol{G})$

Figure 5.18: Rank degeneracy $\mathcal{D}$ evolution of both the Hessian $\boldsymbol{H}$ and GGN $\boldsymbol{G}$ throughout training using the VGG-16 on the CIFAR-100 dataset, the weight $\rho(\lambda)$ corresponds to the spectral mass of the Ritz value $\mathcal{D}(\lambda_0)$ which is closest to the origin.

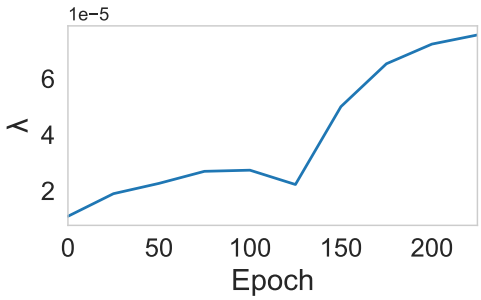
5.9.2 PreResNet110

We repeat the same experiments in Section 5.9.1 for the preactivate residual network with 110 layers, on the same dataset. The slight subtlety is that we can calculate the spectra in both batch normalisation and evaluation mode, which we discuss in Section 5.9.3. Hence we report results for both, with

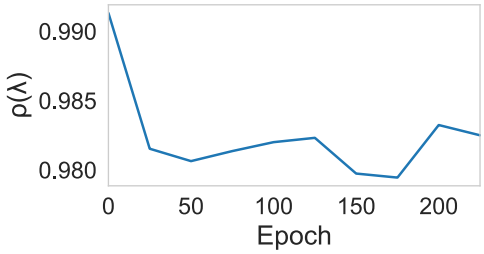
the main finding, that the empirical Hessian spectra are consistent with large rank degeneracy.



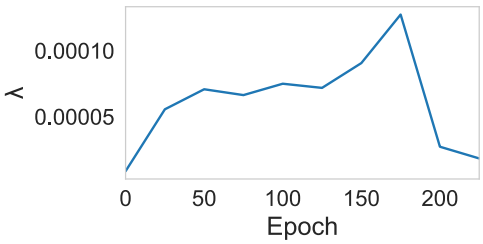
(a) BN train mode Degeneracy



(b) BN train mode Ritz Value



(c) BN eval mode Degeneracy

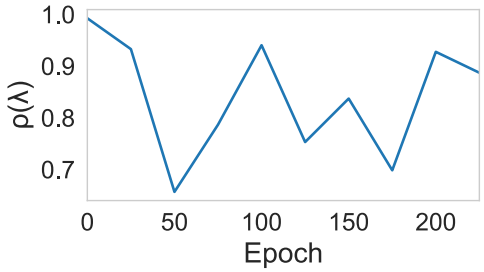


(d) BN eval mode Ritz Value

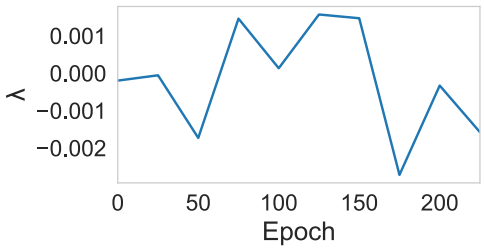
Figure 5.19: Generalised Gauss-Newton rank degeneracy evolution throughout training using the PreResNet-110 on the CIFAR-100 dataset, total training 225 epochs, the Ritz value corresponds to the value of the node which we assign to 0. BNt corresponds to batch normalisation train mode for curvature calculation and BNe for batch normalisation evaluation mode

5.9.3 Subtleties of Batch Normalisation

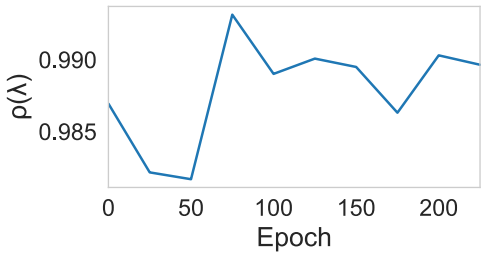
Most modern convolutional neural networks use batch normalisation [Ioffe and Szegedy, 2015]. Whilst we refer the reader to Ioffe and Szegedy [2015] for a full description of batch normalisation, a subtlety which is introduced is that the output of a network for one sample depends on the samples in the mini-batch (as they are used to compute the statistics). Since these statistics are used to normalise and center the layers, they have a large effect on the



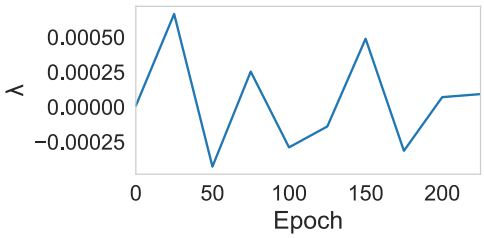
(a) BN train mode Degeneracy



(b) BN train mode Ritz Value



(c) BN eval mode Degeneracy



(d) BN eval mode Ritz Value

Figure 5.20: Hessian rank degeneracy evolution throughout training using the PreResNet-110 on the CIFAR-100 dataset, total training 225 epochs, the Ritz value corresponds to the value of the node which we assign to 0

curvature calculation. In our software package, we give the user two options to deal with this. We denote one **Train Mode** and the other **Evaluation Mode**. We detail them both below

Train Mode: With batch norm in train mode the output of the network for one sample depends on other samples in the mini-batch. This effect influences the result of hessian-vector product computation on a dataset. In particular, if the order of samples in the dataset is changed, then the set of mini-batches also changes and the hessian-vector product will be different. We take this effect into account and fix the order of samples in the dataset, so that hessian-vector product for different networks is computed on the same set of mini-batches. Conceptually, one can use the following interpretation of a neural network with batch norm layers in train mode: the input of

such a network is not a single sample but a mini-batch of samples. Instead of thinking about a dataset of samples we should think about a dataset of mini-batches. With the fixed order of samples, we guarantee that all models are evaluated on the same dataset of mini-batches.

Evaluation Mode: One can also use BN layers in eval mode with BN statistics computed over the whole dataset. With BN layers in eval mode the output of a network on sample does not depend on other samples in the mini-batch. Our code the mode of BN layers can be specified as a parameter for hessian-vector product computation.

5.9.4 Theoretical argument for Feed Forward Networks

For a feed forward neural network, by considering the paths in the network, the input to neuron m_k for a given data point $\boldsymbol{x}$ is given by $\sum_{n_j}^{N_1}\sum_{i}^{d_x} x_i \boldsymbol{w}_{x_i,n_j}\boldsymbol{w}_{n_j,m_k}$, where N_1 is the number of neurons in layer 1. For a neural network $d-1$ hidden layers

$$h_m = \prod_{l=1}^{d-1}\sum_{n_{i,l}=1}^{N_{i,l}}\sum_{i}^{d_x} \boldsymbol{x}_i \boldsymbol{w}_{n_{i,l},n_{i,l+1}}\delta(n_{i,d}=m) \tag{5.38}$$

where $n_{i,l_1} = x_i$ and we fix to the desired output class, through the delta function. The Hessian of the loss in the small loss limit tends to

$$\frac{\partial^2 \ell(h(\boldsymbol{x}_i;\boldsymbol{w}),\boldsymbol{y}_i)}{\partial w_{\phi,\kappa}\partial w_{\theta,\nu}} \to -\sum_{m\neq c}\exp(h_m)\left[\frac{\partial^2 h_m}{\partial w_{\phi,\kappa}\partial w_{\theta,\nu}} + \frac{\partial h_m}{\partial w_{\phi,\kappa}}\frac{\partial h_m}{\partial w_{\theta,\nu}}\right] \tag{5.39}$$

$$\left[\frac{\partial^2 h_m}{\partial w_{\phi,\kappa}\partial w_{\theta,\nu}} + \frac{\partial h_m}{\partial w_{\phi,\kappa}}\frac{\partial h_m}{\partial w_{\theta,\nu}}\right] = \prod_{l=1}^{d-1}\sum_{n_{i,l}\neq[(\phi,\kappa),(\theta,\nu)]}^{N_{i,l}}\sum_{i}^{d_x}\boldsymbol{x}_i\boldsymbol{w}_{n_{i,l},n_{i,l+1}}\delta(n_{i,d}=m)$$
$$+\left(\prod_{l=1}^{d-1}\sum_{n_{i,l}\neq(\theta,\nu)}^{N_{i,l}}\sum_{i}^{d_x}\boldsymbol{x}_i\boldsymbol{w}_{n_{i,l},n_{i,l+1}}\delta(n_{i,d}=m)\right)\left(\prod_{l=1}^{d-1}\sum_{n_{j,l}\neq(\phi,\kappa)}^{N_{j,l}}\sum_{i}^{d_x}\boldsymbol{x}_i\boldsymbol{w}_{n_{j,l},n_{j,l+1}}\delta(n_{j,d}=m)\right) \tag{5.40}$$

Each product of weights contributes an object of rank-1 (as shown in Section 6.3). Furthermore, the rank of a product is the minimum of the constituent ranks, i.e. $\text{rank}(AB) = \min\text{rank}(A,B)$. Hence Equation 5.40 is rank bounded by a $2(\sum_l N_l + d_x)$, where N_l is the total numbers of neurons in the network. By rewriting the loss per-sample and repeating the same arguments and including the class factor

$$\frac{\partial^2 \ell}{\partial w_k \partial w_l} = -\frac{\partial^2 h_{q(i)}}{\partial w_k \partial w_l} + \frac{\sum_j \exp(h_j)\sum_i \exp(h_i)(\frac{\partial^2 h_i}{\partial w_k\partial w_l} + \frac{\partial h_i}{\partial w_k}\frac{\partial h_i}{\partial w_l}) - \sum_i \exp(h_i)\frac{\partial h_i}{\partial w_k}\sum_j \frac{\partial h_j}{\partial w_l}\exp(h_j)}{[\sum_j \exp(h_j)]^2} \tag{5.41}$$

We obtain a rank bound of $4d_y(\sum_l N_l + d_x)$. To give some context, along with a practical application of a real network and dataset, for the CIFAR-10 dataset, the VGG-16 Simonyan and Zisserman [2015] contains 1.6×10^7 parameters, the number of classes is 10 and the total number of neurons is $13,416$ and hence the bound gives us a spectral peak at the origin of at least $1 - \frac{577{,}600}{1.6\times 10^7} = 0.9639$. We give extensive experimental validation for the low-rank nature of the empirical Hessian in Section 5.9.

5.10 Conclusion

This chapter shows under a spiked, field dependent random matrix theory framework that the extremal eigenvalues of the batch Hessian are larger

than those of the empirical Hessian. The magnitude of the perturbation is inversely proportional to the batch size if there are well separated outliers in the Hessian spectrum and inversely proportional to the square-root if not. The main implications of this work are that up until a threshold: 1) SGD learning rates should be scaled linearly with batch size. 2)Adam learning rates should be scaled with the square-root of the batch size. 3) Damping in stochastic second-order methods should be proportional to the ratio of learning rate to batch size. We also note empirically that taking larger steps in the sharp directions of the loss landscape seems to be related to improved generalisation. We validate our predictions and implications extensively on the VGG-16 network and CIFAR-100 dataset, with various hyper-parameter settings, including weight decay and batch-normalisation. For SGD, we further extend validate our prediction on the WideResNet28 $\times$ 10 on the CIFAR-100 and ImageNet-32 datasets. Given that our analysis is neither dataset or architecture specific, we expect our results to hold generally outside of our experimental setup. This work can be used to better inform practitioners of how to adapt learning rate schedules for both small and large devices in a principled manner.

6 | Explaining the Adaptive Generalisation Gap

In the previous chapter we used random matrix theory to consider the implications of mini-batching on the loss surface, with applications in optimisation. The focus of the work was largely on the difference between the batch loss surface and the full dataset loss surface. Whilst we touched on the idea of the true loss surface, we did not explore this in depth. In this Chapter we explore a topic extensively analysed in the field of covariance cleaning, which has been applied in sparse PCA, finance and telecommunications. The idea is that we wish to estimate the covariance matrix under the data generating distribution, however we only have access to a limited number of samples. In the case where we do not have many more samples than the dimension of the matrix we wish to estimate, the empirical estimator of the covariance matrix, is a bad estimate. Random matrix theory can actually be used to account for the deviation from the empirical estimator and the true and hence forms the basis for the "covariance cleaning" recipes. Given that we have in the previous Chapter devised a formalism for the perturbations between the true and empirical loss surfaces, it seems natural to port this thinking into deep learning. In this Chapter we analyse adaptive optimisers, which either explicitly or implicitly model curvature information for enhanced optimisation. We specifically show that from a random matrix theory perspective we expect there to be a generalisation gap (which is observed in practice) and show that many heuristics already employed work by increasing the relative movement in the directions which random matrix theory considers to be less noisy.

6.1 Introduction

The success of deep neural networks across a wide variety of tasks, from speech recognition to image classification, has drawn wide-ranging interest in their optimisation. *Adaptive-Gradient* optimisers, which alter the per parameter learning rate depending on historical gradient information, lead to significantly faster convergence of the training loss than non adaptive methods, such as Stochastic Gradient Descent (SGD) with momentum [Nesterov, 2013]. Popular examples include Adam [Kingma and Ba, 2014], AdaDelta [Zeiler, 2012] and RMSprop [Tieleman and Hinton, 2012]. However, for practical applications the final test set results are more important than the training performance. The difference between training and test performance is known as the *generalisation gap*. For many image and language problems of interest, the test set performance of adaptive-gradient methods is significantly worse than SGD [Wilson et al., 2017]—a phenomenon that we refer to as the *adaptive generalisation gap*. As a consequence of this effect, many state-of-the-art models, especially for image classification datasets such as CIFAR [Yun et al., 2019] and ImageNet [Xie et al., 2019, Cubuk et al., 2019], are still trained using SGD with momentum.

Although less widely used, another class of adaptive methods which suffer from the same phenomenon are *stochastic second-order methods*. These methods, which seek to alter the learning rate along the eigenvectors of the second derivative of the loss function, also suffer from this phenomenon. KFAC uses a Kroenecker factored approximation of the Fisher information matrix (which can be seen as a positive-definite approximation to the Hessian [Martens, 2014]). Other methods use Hessian vector products [Dauphin et al., 2014, Martens, 2010] in conjunction with Lanczos/Conjugate gradients

[Meurant and Strakoš, 2006], which can be seen as a low-rank approximation to the Hessian. Despite strong performance on certain complex tasks such as deep auto-encoders [George et al., 2018], their test performance on large-scale image problems is also lower than that of SGD [Tornstad, 2020].

In this Chapter we argue that adaptive methods are over-confident in their updates in the flattest directions of the loss. We show that this is sub-optimal in terms of optimising the true loss and hence harms generalisation performance. We demonstrate this empirically in an online convex example, where we actively perturb the sharp directions, reducing generalisation without impacting training. We also demonstrate this implicitly for large neural networks by altering the damping/stability constant, which we show alters effective learning rate ratio between the sharp and flat directions.

Given the benefits of adaptive methods, such as swifter convergence and reduced sensitivity to hyper-parameters relative to SGD, understanding the adaptive generalisation gap has significant implications. In this work we show that altering the numerical stability constant in Adam [Choi et al., 2019], can be interpreted as an approximation to linear shrinkage of the noisy Hessian (enabling for reduction of mean square error [Bun et al., 2016]). We further show that many heuristics used to improve generalisation of these methods, such as directly implementing weight decay instead of $L2$ regularisation [Loshchilov and Hutter, 2019], or altering the power in the denominator of the Adam algorithm [Chen and Gu, 2018] can be interpreted within our framework: they correct for the curvature estimation bias and increase relative movement in favour of sharp directions during the optimisation trajectory.

6.1.1 Related Work

Prior theoretical work on generalisation has investigated the correlation between the learning rates to batch size ratio and generalisation during SGD optimisation [Jastrzebski et al., 2020] along with stability analysis to show that larger learning rates lead to lower spectral norms, which under Bayesian and minimum description length arguments should generalise better [Hochreiter and Schmidhuber, 1997]. Li et al. [2019] argue that larger learning rates learn "hard to generalise, easy to fit patterns" better than their lower learning rate counterparts and that this forms part of the generalisation gap. However, to the best of our knowledge, there has been no theoretical work analysing the *adaptive generalisation gap*, with the notable exception of Wilson et al. [2017], who consider adaptive method's poor generalisation performance to be inherent and show this on a simple example.

Practical amendments to improve generalisation of adaptive methods have included: dynamically switch between Adam and SGD [Keskar and Socher, 2017]. Altering the preconditioning matrix in Adam by introducing a free parameter $p \in [0, 1/2]$, where instead of taking the square-root of the second moment of the parameter gradient, the p'th power is taken [Chen and Gu, 2018] . This toggles between Adam and SGD[1]. The choice of $0 \leq p \leq 1/4$ has been shown to have improved convergence properties to that of AMSgrad [Zhou et al., 2018]. Another recent popular amendment has been implementing weight decay instead of $L2$ regularisation, which are not equivalent for adaptive methods [Loshchilov and Hutter, 2019]. Zhang et al. [2018], investigate the effect of implementing decoupled weight decay for KFAC and similarly note increased generalisation performance. Choi et al. [2019] shown

[1] $p = 1/2$ reduces to Adam and $p = 0$ reduces to SGD with learning rate $\alpha/1 + \epsilon$.

that by altering the damping or numerical stability constant, typically taken as 10^{-8} in Adam, Adam can simulate SGD and retain its generalisation performance.

While empirically effective, these alterations lack a clear theoretical motivation. Neither Keskar and Socher [2017], Choi et al. [2019], Loshchilov and Hutter [2019] nor Chen and Gu [2018] explain why Adam (or adaptive methods in general) inherently generalises worse than SGD. Instead, they show that switching from Adam to SGD, or making Adam more similar to SGD brings improvements. A clear analysis of how SGD differs from adaptive methods and why this impacts generalisation is not provided in any of the aforementioned works.

6.1.2 Contributions

In this chapter we conjecture that a key driver of the adaptive generalisation gap is that adaptive methods *fail to account for the greater levels of noise associated with their estimates of flat directions in the loss landscape.* What this essentially means is that because of the $1/\lambda_i$ scaling in second order methods (and by proxy adaptive methods also), that much of the optimisation trajectory is devoted to moving in directions specific to a given minibatch and not inherent to the underlying true loss surface (which would correspond to the best generalisation performance). The fundamental principle underpinning this conjecture—that sharp directions contain information from the underlying process and that flat directions are largely dominated by noise—is theoretically motivated from the spiked covariance model [Baik and Silverstein, 2004]. This model has been successfully applied in Principal Component Analysis (PCA), covariance matrix estimation and finance [Bloemendal et al., 2016, Everson and Roberts, 2000, Bun et al., 2017]. We revisit

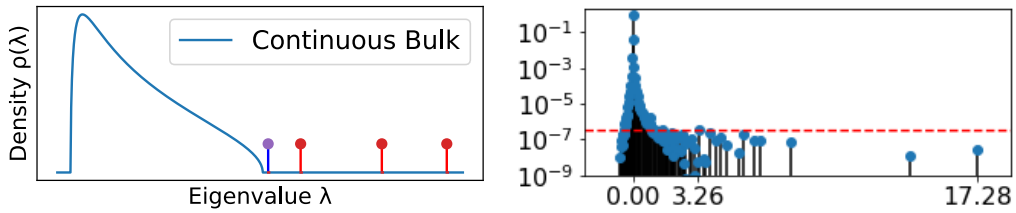
this idea in the context of deep neural network optimisation by defining the *estimated curvature learning rate ratio*:

$$\mathcal{R}_{\text{est-curv}} := \frac{\alpha_{\text{flat}}}{\alpha_{\text{sharp}}} \tag{6.1}$$

where α_{flat} and α_{sharp} are the learning rates along the flat and sharp directions, respectively and this ratio encapsulates the noise-to-signal ratio as motivated by our conjecture. We show first that by increasing $\mathcal{R}_{\text{est-curv}}$ on both logistic regression with the MNIST dataset and Deep Neural Networks such as the VGG-16 on the CIFAR-100 dataset, we can decrease generalisation performance whilst minimally impacting training.

Our results indicate that typical hyper-parameter settings for adaptive methods produce a systematic bias in favour of an $\mathcal{R}_{\text{est-curv}}$ that is too large and that this is a contributing factor in the adaptive generalisation gap. We show that increasing the damping or numerical stability constants in adaptive methods can be seen as an amalgamation of learning rate reduction and linear shrinkage of the implicit Hessian/covariance of gradients matrix. This shows that the practice of *epsilon tuning* Choi et al. [2019] is theoretically justified and should be employed in typical optimisation routines. Our framework further provides theoretical justification for modern developments in improving generalisation of adaptive algorithms, namely *decoupled weight decay* [Loshchilov and Hutter, 2019] and *partial adaptivity* [Chen and Gu, 2018]. Each of these methods effectively decreases $\mathcal{R}_{\text{est-curv}}$, correcting for the bias.

Chapter Structure: In Section 6.2 below, we first describe the theory that motivates our generalisation conjecture. In Sec. 6.3 we introduce adap-



(a) Hypothetical $\rho(\lambda)$ (b) VGG-16 C-100 Hessian

Figure 6.1: **Spiked random matrix model, in theory (a) and a neural network spectrum in practice (b)**(a) shows a Hypothetical spectral density plot where we have a continuous bulk with sharp support, a finite size fluctuation (shown in blue) which corresponds to the Tracy-Widom region and three well-separated outliers (shown in red). (b) VGG-16 Hessian $\boldsymbol{H} \in \mathbb{R}^{P \times P}$ on the CIFAR-100 dataset at epoch 150. The red dotted line corresponds to κ/P, where $P = 1.6 \times 10^7$ and $\kappa = 5$, which is taken as an approximate threshold defining the boundary of the bulk.

tive optimisers and how our conjecture relates to their generalisation performance. We provide empirical evidence to support our claims in Sections 6.4 and 6.5. Finally, we relate our conjecture to several widely-used optimisation techniques in Sec. 6.6.

6.2 Spiked Model, Outliers, and the True Loss

Covariance cleaning considers the problem of how a covariance matrix $\boldsymbol{H} \in \mathbb{R}^{P \times P}$ under the expectation the data generating distribution (true covariance matrix) differs from that of its empirical average using N samples (the empirical covariance matrix). In the limit $N \to \infty$ for finite P, by the central limit theorem the empirical estimator converges to the true. However when $q = P/N > 0$, the deviations can be significant [Bun et al., 2017]. The spectral perturbations due to this limited sampling can be characterised analytically by assuming the perturbation to be the addition or multiplication of a random matrix [Benaych-Georges and Nadakuditi, 2011], which we refer to

as the "noise" matrix. The difference between the empirical covariance matrix and the true covariance matrix is entirely dependent on the interaction between the eigenvalue/eigenvector pairs of the true covariance matrix with those of the fluctuations matrix. Many adaptive-gradient methods form an online approximation to the gradient covariance and hence these perturbations are relevant to their generalisation. For stochastic second-order methods, Chapter 5 extended these ideas to formally include the Hessian of neural networks. In the next Section we argue that the eigenvectors corresponding to the eigenvalues of largest magnitude are least affected and hence most accurately estimated.

6.2.1 Intuition: Why Sharp directions retain more information than Flat ones

The spectrum of the fluctuations matrix occupies a continuous region (sharp in the asymptotic limit [Bun et al., 2017]) as shown in Figure 6.1a, known as the "bulk" supported between $[\lambda_-, \lambda_+]$ [Bun et al., 2017, 2016]. A bulk region has also been observed in deep learning [Granziol et al., 2019b, Papyan, 2019, Sagun et al., 2018]. Within this "bulk" all the information about the original eigenvalue/eigenvector pairs is lost [Baik et al., 2005]. These eigenvectors are uniformly distributed on the unit sphere [Benaych-Georges and Nadakuditi, 2011]. We therefore define any eigenvalue $\lambda_i \leq \lambda_+$ as "flat". For finite-size samples and network size, there exists a region beyond the predicted asymptotic support of the fluctuations matrix, called the Tracy-Widom region [Tracy and Widom, 1994, El Karoui et al., 2007], where there may be isolated eigenvalues which are part of the fluctuations matrix spectrum (shown in the Figure). Anything beyond the Tracy-Widom region $\lambda_i \gg \lambda_+$ (typically in the literature only semi-positive-definite matrices are consid-

ered, but it is easy to extend the arguments to negative outliers $\lambda_i \ll \lambda_-$), is considered an outlier and corresponds to a "sharp" direction. *Such directions represent underlying structure from the true data.* The eigenvectors corresponding to these eigenvalues can be shown to lie in a cone around their true values [Benaych-Georges and Nadakuditi, 2011]. In Figure 6.1b, we show the Hessian at the 150'th epoch of the VGG-16. Here, similar to our hypothetical example, we see a continuous region, followed by a number of eigenvalues which are close to (but not within) the bulk, and finally, several clear outliers.

6.2.2 Key Result

We highlight a particular insight from Chapter 5, with the full proof in Benaych-Georges and Nadakuditi [2011]. However, instead of considering the empirical loss, we look at the True Loss.

Theorem 17. *The eigenvector overlap between the eigenvectors* $\hat{\boldsymbol{\phi}}_i \in \mathbb{R}^{P\times 1}$ *of the Hessian of the mini-batch* $\nabla^2 L_{batch}(\boldsymbol{w}_k)$ *of batch size* B *and the eigenvectors* $\boldsymbol{\phi}_i$ *of the Hessian* $\nabla^2 L_{true}(\boldsymbol{w}_k)$ *under the expectation of the true data distribution* $\phi_{\boldsymbol{x},\boldsymbol{y}}$ *, where* $L_{true}(\boldsymbol{w}_k) = \int \ell(\boldsymbol{w}_k; \boldsymbol{x}, \boldsymbol{y}) d\phi_{\boldsymbol{x},\boldsymbol{y}}$ *are given by*

$$|\hat{\boldsymbol{\phi}}_i^T \boldsymbol{\phi}_i|^2 = \begin{cases} 1 - \frac{P\sigma^2}{B\lambda_i^2} & \text{if } |\lambda_i| > \sqrt{\frac{P}{B}}\sigma, \\ 0 & \text{otherwise,} \end{cases} \tag{6.2}$$

where σ captures the sampling noise per Hessian element for a single sample.

To summarise: the key takeaway of this result is that *sharper directions possess a better signal-to-noise ratio than flatter directions in estimating the true loss surface.* This effect is important for adaptive methods which rely on

curvature estimation (either explicitly for stochastic second-order methods or implicitly for adaptive-gradient methods) to accelerate their progress on the true loss surface.

In the next Section, we provide empirical evidence that as we increase the *estimated curvature learning rate ratio* in the flat-to-sharp directions $\mathcal{R}_{\text{est-curv}}$, generalisation indeed suffers (lending $\mathcal{R}_{\text{est-curv}}$ a loose interpretation as a noise-to-signal ratio). It is interesting to note that in the original Adam paper Kingma and Ba [2014], the authors define the square-root of their diagonal approximation to the covariance of gradients matrix as "noise". Our work suggests that this does not quite capture the full picture—a further correction term is required to prevent adaptive-gradient methods from exhibiting a bias towards a greater $\mathcal{R}_{\text{est-curv}}$ value than desired. We quantify this statement, and its relationship to linear shrinkage, more precisely in Sec. 6.3.2. In fact we experimentally show in Sec. 6.5.2 that by increasing the movement along along precisely these high variance which directions they define as "noise", that generalisation performance improves even beyond that of SGD. This lends credence to the interpretation that these high variance directions instead constitute signal.

6.3 Adaptive Optimisation

Consider a general iterative optimiser that seeks to minimise the scalar loss $L(\boldsymbol{w})$ for a set of model parameters $\boldsymbol{w} \in \mathbb{R}^P$. The $k+1$-th iteration of such an optimiser can be written[2] as follows:

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha \boldsymbol{B}^{-1} \nabla L_{batch}(\boldsymbol{w}_k) \tag{6.3}$$

[2]Ignoring additional features such as momentum and explicit regularisations.

where α is the global learning rate. For SGD, $\boldsymbol{B} = \boldsymbol{I}$ whereas for adaptive methods, $\boldsymbol{B}$ typically provides some form of approximation to the Hessian $\boldsymbol{B} \approx \nabla^2 L_{batch}(\boldsymbol{w}_k)$.

To take a prominent example, Adam uses the square-root of the moving uncentred second moment of the per-parameter gradient [Kingma and Ba, 2014] that can be interpreted as a running diagonal approximation to the covariance of gradients. It has been argued that this is close to the Hessian for DNNs [Zhang et al., 2019a, Jastrzebski et al., 2020, Fort et al., 2019, Liu et al., 2019, He and Su, 2019]. Other methods use Hessian vector products, using either the Lanczos or conjugate gradient techniques [Meurant and Strakoš, 2006] to approximate $\boldsymbol{B}^{-1}\nabla L_{batch}(\boldsymbol{w}_k)$ [Martens, 2010, Dauphin et al., 2014] (which for a number of products $m \ll P$ can be seen as a low-rank inverse approximation to $\boldsymbol{B}$) or direct inversion of Kronecker factored approximations [Martens and Grosse, 2015].

Writing this update in the eigenbasis of the Hessian[3] $\boldsymbol{H} = \nabla^2 L_{batch}(\boldsymbol{w}_k) = \sum_i^P \lambda_i \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T \in \mathbb{R}^{P\times P}$, where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_P \geq 0$ represent the ordered scalar eigenvalues, the parameter step takes the form:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \sum_{i=1}^{P} \frac{\alpha}{\lambda_i + \delta} \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T \nabla L(\boldsymbol{w}_k). \tag{6.4}$$

Here, δ is a damping (or numerical stability) term. This damping term, which is typically grid searched [Dauphin et al., 2014] or adapted during training [Martens and Grosse, 2015], can be interpreted as a trust region [Dauphin et al., 2014] that is required to stop the optimiser moving too far in directions deemed flat ($\lambda_i \approx 0$) and diverging. In Adam Kingma and Ba [2014], it is set

[3] which we either assume to be positive-definite or that we are working with a positive-definite approximation thereof

to 10^{-8}. For small values of δ, α must also be small to avoid optimisation instability, hence global learning rates and damping are coupled in adaptive optimisers.

6.3.1 Adaptive updates, damping and the estimated curvature learning rate ratio

The learning rate in the flattest ($\lambda \approx 0$) directions is approximately $\frac{\alpha}{\delta}$, which is larger than that in the sharpest ($\lambda_i \gg \delta$) directions $\frac{\alpha}{\delta+\lambda_i}$. This difference in per direction effective learning rate makes the best possible loss reduction under the assumption that the loss function can be effectively modelled by a quadratic. Crucially, however, it does not account for how accurately each eigenvector component of the update estimates the true underlying loss surface, which is described in Theorem 17. Ignoring the gradient overlap with the eigenvectors $\boldsymbol{\phi}_i \nabla L(\boldsymbol{w}_k)$ and assuming that the smallest eigenvalue $\lambda_P \ll \delta$, we see that $\mathcal{R}_{\text{est-curv}} = 1 + \frac{\lambda_1 - \lambda_P}{\delta}$. This is in contrast to SGD where $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \sum_{i=1}^{P} \alpha \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T \nabla L(\boldsymbol{w}_k)$ and hence $\mathcal{R}_{\text{est-curv}} = 1$.

The crucial point to note here is that the difference in $\mathcal{R}_{\text{est-curv}}$ is primarily controlled by the damping parameter: smaller values yield a larger $\mathcal{R}_{\text{est-curv}}$, skewing the parameter updates towards flatter directions. From Theorem 17 we see that overlap is reduced proportionally to the number of network parameters P and inversely to the number of samples taken for evaluation B, and so we expect the influence of this effect to become more significant for larger models (we show this to be the case in Sec. 6.5.1).

In the next Section we show that the damping parameter δ can be decomposed into a learning rate reduction term and a linear shrinkage of the curvature matrix.

6.3.2 Damping as an Approximate Linear Shrinkage

By re-writing the scaling applied in the direction of the i^{th} eigenvector, we note that:

$$\frac{1}{\lambda_i + \delta} = \frac{1}{\kappa(\beta\lambda_i + (1-\beta))} \tag{6.5}$$

Solving gives us $\kappa = \frac{1}{\beta}, \beta = \frac{1}{1+\delta}$. Hence, using a damping δ is equivalent to using a learning rate of $\frac{\alpha}{\delta+1}$ and applying a linear shrinkage factor $\frac{1}{1+\delta}$ to the Hessian estimation. Thus larger damping corresponds to more aggressive shrinkage and hence the practice of *epsilon tuning* [Choi et al., 2019] can be viewed as approximate linear shrinkage. This idea that a combined estimator (of which linear shrinkage is a type) can outperform the sample estimator is known as "Stein's Phenomenon" [Stein, 1956].

Optimality of Linear Shrinkage: Bun et al. [2016] show that under the free multiplicative noise model, the linear shrinkage estimator

$$\tilde{\boldsymbol{H}} = \beta\boldsymbol{H} + (1-\beta)\boldsymbol{I} = \arg\min_{\boldsymbol{H}*} ||\boldsymbol{H}* - \boldsymbol{H}_{true}||_{L2}$$

gives the minimum error between the estimator and the true covariance matrix. This implies that with an appropriately chosen shrinkage coefficient the optimiser more accurately descends the directions of the true risk. We expect such an optimiser to perform better in validation and test set performance, which is the case in our experiments in Section 6.5. In practice we are not correcting the incorrect eigenvector directions, but simply the eigenvalues associated with those vectors. We note that the practice of leaving the estimated eigenvectors unperturbed but shrinking the eigenspectrum, is well established and successfully applied in the field of sparse component analysis and finance [Bun et al., 2017].

6.4 Toy Example: MNIST

We validate our conjecture that movements in the sharp direction of the loss landscape are vital to generalisation, we first run a convex non stochastic example where we both explicitly and implicitly perturb the movement along the sharpest directions of the loss.

We implement a second-order optimiser based on the Lancozs iterative algorithm [Meurant and Strakoš, 2006] (LanczosOPT). As a baseline we also implement gradient descent (GD). We employ a training set of 1K MNIST [LeCun, 1998] examples using Logistic Regression and validate on a held out test set of 10K examples.

6.4.1 LanczosOPT for Logistic Regression

The Lanczos Algorithm is an iterative algorithm for learning a subset of the eigenvalues/eigenvectors of any Hermitian matrix, requiring only matrix vector products. When the number of Lanczos steps, m, is significantly larger than the number of outliers, the outliers in the spectrum are well learned and estimated Granziol et al. [2019b]. Since the number of well-separated outliers from the spectral bulk is at most the number of classes [Papyan, 2019] (which is $n_c = 10$ for this dataset), we expect that Lanczos algorithm to pick out these well-separated outliers when the number of iterations $k \gg n_c$ [Granziol et al., 2019b, Meurant and Strakoš, 2006] and therefore use $k = 50$. To investigate the impact of scaling steps in the Krylov subspace given by the sharpest directions, we consider the update $\boldsymbol{w}_{k+1}$ of the form:

$$\boldsymbol{w}_k - \alpha\left(\sum_i^k \frac{1}{\lambda_i + \delta}\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T\nabla L(\boldsymbol{w}_k) + \sum_{k+1}^P \frac{1}{\delta}\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T\nabla L(\boldsymbol{w}_k)\right) \tag{6.6}$$

where $P = 7850$ (the number of model parameters) and hence the vast majority of "flat" directions remain unperturbed. For fixed α, δ controls the estimated curvature learning rate ratio.

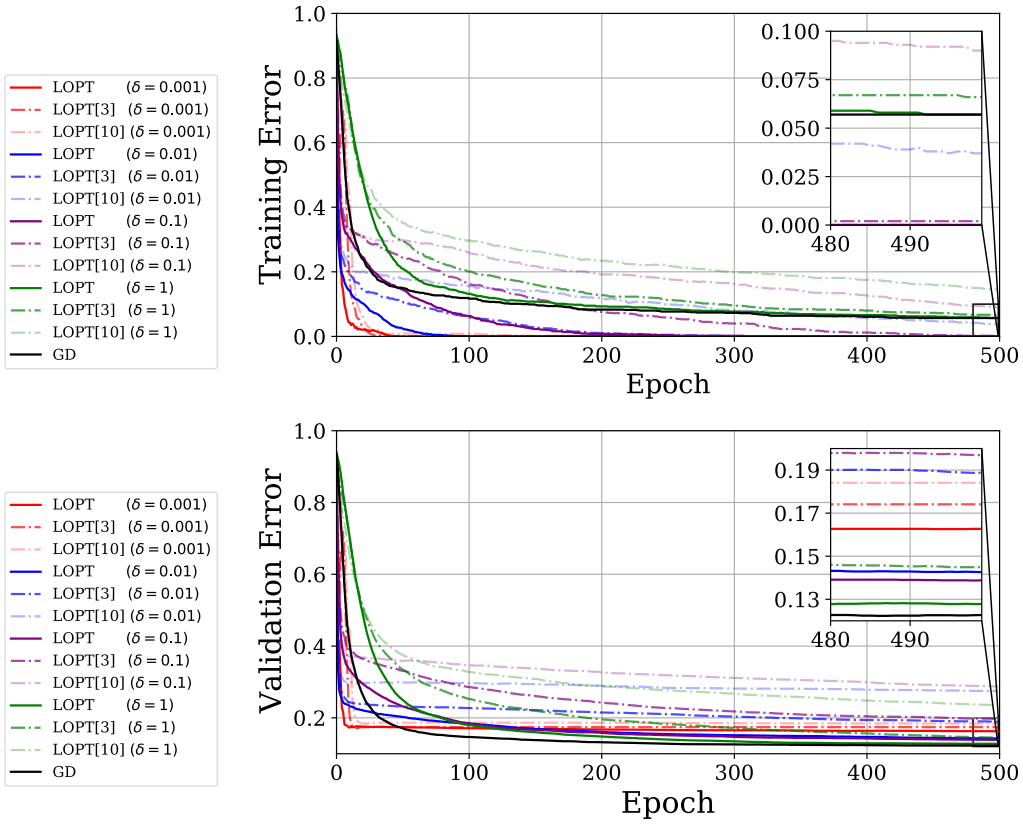
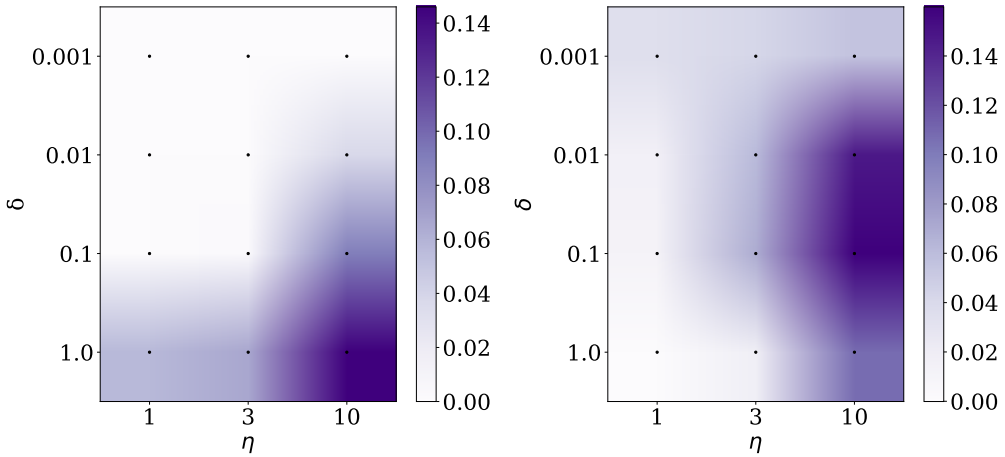


Figure 6.2: Training/Test Error of LanczosOPT/Gradient Descent (LOPT/GD) optimisers for Logistic Regression on the MNIST dataset with fixed learning rate $\alpha = 0.01$ across different damping values, δ. LOPT[η] denotes a modification to the LOPT that perturbs a subset of update directions by a factor of η (see Sec. 6.4.1 for details).

Experimental Results: In Fig. 6.2, we observe evidence consistent with our central hypothesis: namely that as we increase $\mathcal{R}_{\text{est-curv}}$ (by decreasing the value of δ for a fixed α value of 0.01), the generalisation of the model suffers correspondingly.

To explore this effect in more detail, we introduce perturbations to the optimiser (denoted LOPT[η]), in which we scale up the eigenvalues in the denominator of Eqn. 6.6 by a factor of η (we explore scaling factors of 3 and 10). This explicitly reduces movement in sharp directions and consequently increases reliance on flat directions (which are left unperturbed). For each fixed value of δ, we see clearly that perturbations of greater magnitude cause greater harm to generalisation. Another interesting phenomenon which we note, is that for larger values of δ the perturbed optimisers suffer more gravely in terms of the effect on both training and validation. We visualise this in heat map form in Fig. 6.4, where we show the difference from the best training and testing error as a function of δ, η. We note that the generalisation of all algorithms is worsened by explicit limitation of movement in the sharp directions (and an increase of estimated curvature learning rate ratio), however for extremely low damping measures (which are typical in adaptive optimiser settings) there is no or very minimal impact in training performance (upper region of Fig. 6.4 (a)).



(a) $\Delta(\delta, \eta)$ Training (b) $\Delta(\delta, \eta)$ Testing

Figure 6.4: Error change with damping/sharp direction perturbation δ, η in LanczosOPT. Best run error set to 0. Darker regions indicate higher error.

6.5 Neural Networks: CIFAR100

To explore the conjecture for modern deep learning architectures for which running a large number of matrix vector products is infeasible, we first run the second-order optimiser KFAC [Martens and Grosse, 2015] and then popular adaptive-gradient optimiser Adam [Kingma and Ba, 2014] on the VGG-16 on the CIFAR-100 dataset. We specifically choose this network, as despite its size (over 16 million parameters) it can be effectively trained without batch-normalisation and weight decay hence allows us to isolate the effect of $\mathcal{R}_{\text{est-curv}}$, as opposed to the effect of different regularisation implementations for adaptive and non-adaptive methods as discussed in Loshchilov and Hutter [2019], Zhang et al. [2018].

Both from a theoretical and practical perspective, it is beneficial to decay the learning rate. Whilst optimal asymptotic convergence guarantees are given for learning rates which decay proportionally to the square-root of the number of iterations, or alternatively stay flat with a step size proportional to the inverse square-root of the iteration number [Duchi, 2018, Nesterov, 2013], such aggressive decay rates or low learning rates are rarely employed in practice. Practitioners typically adopt a wide variety of schedules, with step scheduling amongst the most popular, although cosine annealing is also widely used [Loshchilov and Hutter, 2017] . Even though adaptive optimisation methods are less sensitive to the scheduling, convergence proofs, require a learning rate reduction Reddi et al. [2018] and in practice the performance without scheduling is significantly hampered. We use a Linear schedule (detailed in suppl. material).

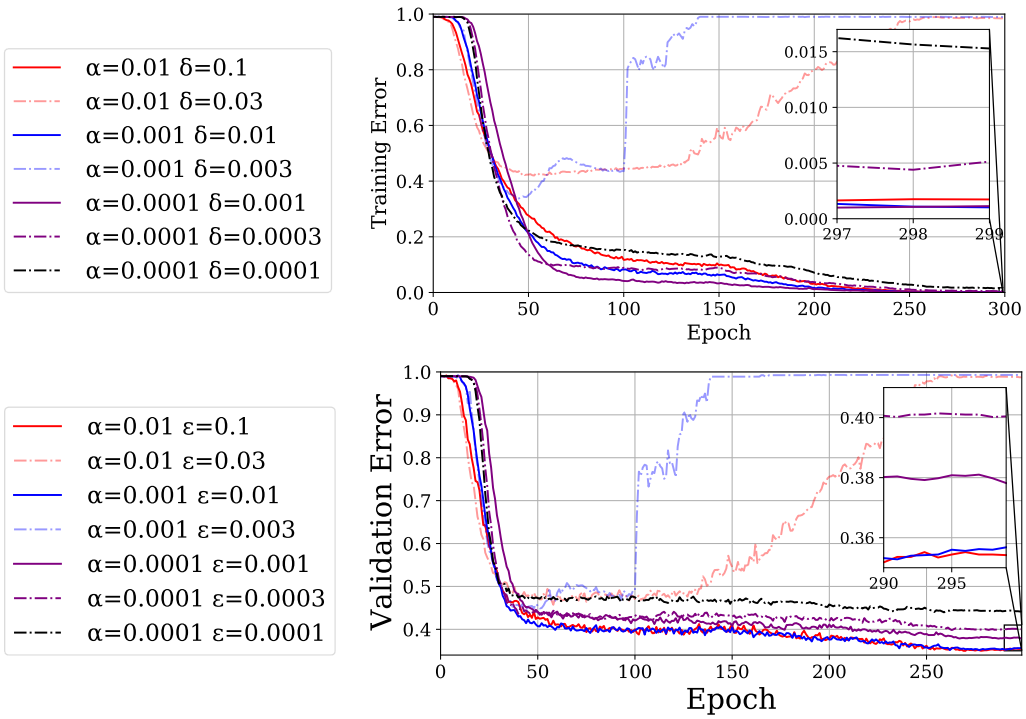


Figure 6.5: **Second order stochastic optimisers with larger learning rates and larger damping generalise better:** Training/Validation Error of the KFAC optimiser for the VGG-16 on the CIFAR-100 dataset with various learning rates α and damping values, δ.

6.5.1 KFAC VGG-16 CIFAR-100

By decreasing the global learning rate α whilst keeping the damping to learning rate ratio $\kappa = \frac{\delta}{\alpha}$ constant, we reduce the estimated curvature learning rate ratio, $\mathcal{R}_{\text{est-curv}}$, which is determined by $\frac{\lambda_i}{\kappa\alpha}+1$. In Fig. 6.5 we observe that as we increase $\mathcal{R}_{\text{est-curv}}$ the training performance is effectively unchanged, but generalisation suffers (65% $\rightarrow$ 62.2%). Whilst decreasing the damping results in poor training for large learning rates (divergent trajectories in Fig. 6.5), for very low learning rates the network efficiently trains with a lower damping coefficient. Such regimes further increase $\mathcal{R}_{\text{est-curv}}$ and we observe that they generalise more poorly. For $\alpha = 0.0001$ dropping the damping coefficient δ

from to 0.0003 and 0.0001 drops the generalisation further to 60.2% and then 56% respectively. Similarly to Logistic Regression, for both cases the drop in generalisation is significantly larger than the drop in training accuracy.

6.5.2 Adam VGG-16 CIFAR-100

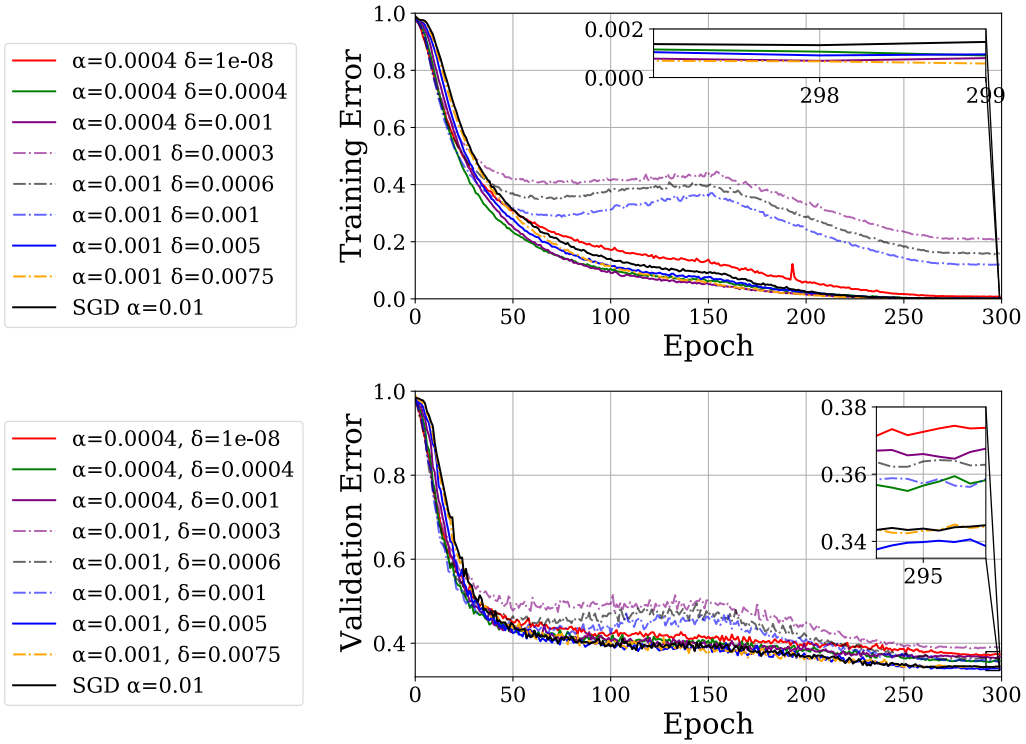


Figure 6.6: **Adaptive optimisers with larger learning rates and larger damping generalise better:** Training/Validation Error of the Adam optimiser for the VGG-16 on the CIFAR-100 dataset with various learning rates α and damping values, δ.

We also run the Adam optimiser on the same network and dataset for a variety of learning rate and damping coefficients as shown in Figure 6.6 and plot the results against the SGD baseline with maximal stable learning rate $\alpha = 0.01$ (corresponding to optimal performance). For the largest learning rate with which Adam trains $\alpha = 0.0004$ with the standard damping coeffi-

cient $\delta = 10^{-8}$, we see that Adam signficantly underperforms SGD, but that this gap is reduced by simply increasing damping coefficient without harming performance. Over-damping decreases the performance.

For larger global learning rates enabled by a significantly larger than default damping parameter, when the damping is set too low, the training is unstable (corresponding to the dotted lines). None the less many of these curves with poor training out-perform the traditional setting on testing. We find that for larger damping coefficients $\delta = 0.005, 0.0075$ Adam is able to match or even beat the SGD baseline, whilst converging faster. This evidences that for real problems of interest, adaptive methods may not be inherently worse than their non-adaptive counterparts as argued by Wilson et al. [2017].We note as shown in Table 6.1, that whilst decreasing δ always leads to smaller spectral norm, this does not always coincide with better generalisation performance.

To investigate whether our alteration of δ could useful for deep learning practitioners, We run an experiment on the VGG-16 using both batch normalisation [Ioffe and Szegedy, 2015] and decoupled weight decay [Loshchilov and Hutter, 2019]. We use a learning rate of 0.001 and a decoupled weight decay of $[0, 0.25]$. For this experiment (see suppl. material for training and validation curves) we note that using a larger damping constant slightly assists training and improves generalisation, both without weight decay and with weight decay. This indicates that reducing $\mathcal{R}_{\text{est-curv}}$ using something as simple as increasing the damping constant δ could be beneficial in practice Choi et al. [2019]. The spectral perturbation from the True Hessian (shown in Theorem 17) is a function of the sampling noise. This is estimated in Chapter 5 and can be shown to change during training. Hence potentially future work could consider scheduling the damping constant δ for even further performance improvements.

6.5.3 What about Batch Normalisation?

For simplicity in the previous set of experiments, we did not use batch normalisation. It is curious to then ask if such observed performance improvements hold in more practical setups where weight decay and batch normalisation are employed. We show on that VGG-16 in Figure 6.7 that a similar degree of improvement holds. Furthermore we see on the ResNet-50 on the ImageNet dataset, a typical benchmark that by increasing the damping coefficient to equal the learning rate $\delta = 0.0001$ we match the final SGD testing accuracy (whilst converging much faster), we also note that by increasing the epsilon coefficient further (and also increasing the learning rate from $0.001 \rightarrow 0.003$) that we can even surpass this SGD baseline test performance.

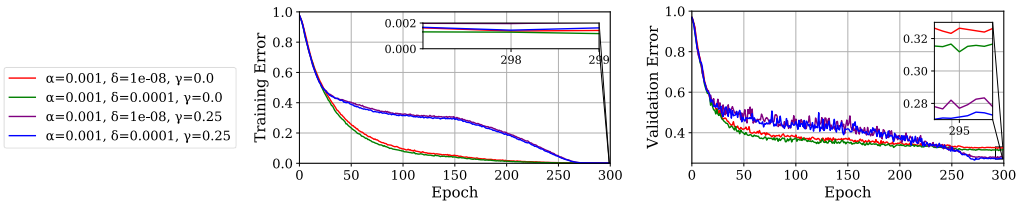


Figure 6.7: **Adam can outperform SGD with epsilon tuning on a VGG:** Training/Validation Error of the Adam optimiser for the VGG-16 using Batch Normalisation and Decoupled Weight Decay on the CIFAR-100 dataset with various learning rates α and damping values, δ.

Link to Solution Sharpness: Motivated by the literature on sharpness and generalisation from a Bayesian and minimum description length perspective [Hochreiter and Schmidhuber, 1997] and the link between flatness and learning rates [Jastrzkbski et al., 2017, Wu et al., 2018], we investigate whether increasing $\mathcal{R}_{\text{est-curv}}$ is related to solution sharpness and show the results in Fig. 6.2. We see that increasing $\mathcal{R}_{\text{est-curv}}$ increases solution sharp-

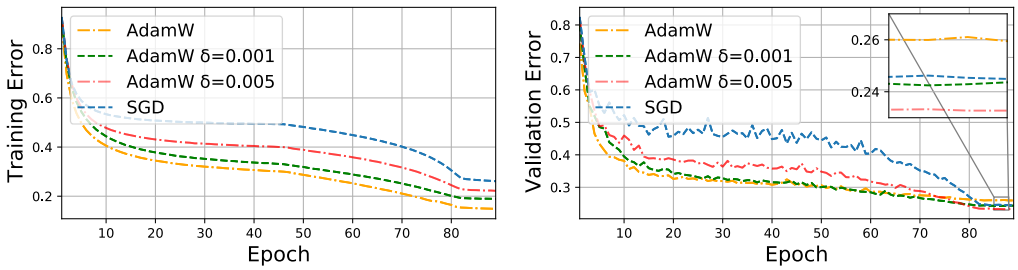


Figure 6.8: **Adam can outperform SGD with epsilon tuning on a ResNet:** Training/Validation Error of the AdamW optimiser for the ResNet-50 on the ImageNet dataset with various learning rates $\alpha = 0.001, 0.003$ and damping values, δ against an SGD baseline.

$\delta\downarrow, \alpha\rightarrow$	0.0004	0.001
0.0000001	**53.07**(62.85)	
0.0004	**21.14**(64.5)	
0.001	**9.94**(63.54)	**20.8**(64.38)
0.005		**9.07**(66.24)
0.0075		**2.37**(65.75)

Table 6.1: Adam

$\delta\downarrow, \alpha\rightarrow$	0.001	0.0001
0.1	**6.65**(65.02)**	
0.01	**20.8**(64.84)	
0.001		**48.2**(62.18)
0.0003		**527.2**(60.18)
0.0001		**711.3**(56.04)

Table 6.2: KFAC

Table 6.3: Spectral norm λ_1 at training end, in **bold**, with corresponding (validation accuracy). For learning rate/damping α, δ on CIFAR-100 using KFAC/Adam on the VGG-16 **($\alpha = 0.1$)

ness and decreases generalisation. As can be seen from the results where $\alpha = 0.0001$, this result is very prominent even for a fixed learning rate. The results of this experiment suggest that $\mathcal{R}_{\text{est-curv}}$ (rather than α alone [Wu et al., 2018]) has a significant influence on sharpness and generalisation in second-order methods. Our experiments on this network also confirm that the adaptive generalisation gap is prevalent even without alternative implementations of weight decay [Loshchilov and Hutter, 2019, Zhang et al., 2018] and hence this also is not the sole source of the adaptive generalisation gap. In fact as we shown in Section 6.6, decoupled weight decay in combination

with batch normalisation increases the movement along the sharpest directions, reducing $\mathcal{R}_{\text{est-curv}}$.

6.6 Prior work reduces reliance on flat directions.

In this section, we discuss how several recently introduced techniques for improving optimisation may relate to our conjecture. Let us consider the update equations for both SGD and Adaptive methods in the basis of the eigenvectors of the Hessian. We assume that the pre-conditioning matrix $\boldsymbol{B} \approx \boldsymbol{H}$. We also assume that $L2$ regularisation of magnitude γ is employed. The $\boldsymbol{w}_{k+1}$'th weight vector is then given by:

$$\begin{cases} (1-\gamma\alpha)\sum_i^P \boldsymbol{\phi}_i\boldsymbol{\phi}_i^T\boldsymbol{w}_k - \alpha\sum_i^P \boldsymbol{\phi}_i^T\nabla L(\boldsymbol{w})\boldsymbol{\phi}_i & \text{SGD} \\ \sum_i^P(1-\frac{\gamma\alpha}{\lambda_i+\delta})\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T\boldsymbol{w}_k - \alpha\sum_i^P \frac{\boldsymbol{\phi}_i^T\nabla L(\boldsymbol{w})}{\lambda_i+\delta}\boldsymbol{\phi}_i & \text{Adam} \end{cases} \tag{6.7}$$

where the damping δ is set to 10^{-8} in Adam. *Decoupled weight decay* [Loshchilov and Hutter, 2019] alters Adam such that the weight decay update term[4] in Eqn 6.7 resembles that of SGD instead of that of Adam. This increases the weight decay along the sharpest directions of the loss which in turn increases the estimated curvature learning rate ratio in favour of these directions. To demonstrate this point, consider the difference in channel weight vector direction $\hat{\boldsymbol{w}}_k$ [Hoffer et al., 2018] in the basis of the sharpest eigenvectors after one update of SGD:

$$\sum_i^P \boldsymbol{\phi}_i\boldsymbol{\phi}_i^T\left(\hat{\boldsymbol{w}}_{k+1}-\hat{\boldsymbol{w}}_k\right) = \frac{-\eta(\boldsymbol{I}-\sum_i^P \boldsymbol{\phi}_i\hat{\boldsymbol{w}}_k{}^T\nabla L(\hat{\boldsymbol{w}}_k)\boldsymbol{\phi}_i^T\hat{\boldsymbol{w}}_k)}{||\boldsymbol{w}_k||^2}. \tag{6.8}$$

[4]Colour is used to highlight the relevant terms in the formula.

Hence decoupled weight decay, which by the first term in Equation 6.7 reduces the component along $\boldsymbol{\phi}_i^T \hat{\boldsymbol{w}}_k$, through the reduction of the corrective term on the RHS of Eqn. 6.8, increases the effective learning rate. In Padam [Chen and Gu, 2018] by changing the square-root in the Adam update to an arbitrary power $p < 1/2$, increases both the weight decay and the update in the sharp directions in Equation 6.7), increasing the effective learning rate. Epsilon tuning [Choi et al., 2019], by increasing both ϵ and α increases both the weight decay term and update in the direction of the sharpest eigenvectors term.

6.6.1 Comment on Data

Although we note that our technical assumptions do not specify any particular choice of data and we expect our results hence to hold generally, our experiments rely on CIFAR-100 and ImageNet which is a relatively "clean" dataset. We note firstly that even on these clean datasets, it is not possible to overfit to the training set if the model is of sufficiently low complexity, such as Logistic regression. This is partially captured in our formulae through the dependence on the number of parameters P which serves as a crude measure of complexity. In such cases, these discussions are not overly helpful in terms of characterising the adaptive generalisation gap as there is no generalisation gap and hence no adaptive generalisation gap. It may also be that the dataset is corrupted and hence we have fundamental error that perturbs the training data in such a way that the assumption of the data being sampled i.i.d from the data generating distribution is violated rendering the corresponding analysis irrelevant.

6.7 Conclusion

In this Chapter we show using the spiked covariance model that we expect sharp directions of the sampled loss surface to retain more information about the true loss surface compared to flatter directions. We show that for adaptive methods, which attempt to minimise the local quadratic of the sampled loss surface, that this leads to sub-optimal steps with worse generalisation performance. As a consequence, our framework gives a theoretical interpretation to many tried and tested techniques to improve adaptive-gradient methods, such as decoupled weight decay, epsilon tuning and partial adaptivity, which we show are all methods which increase the movement along the sharpest directions in the loss surface. We further investigate the effect of damping on the solution sharpness and find that increasing damping always decreases the solution sharpness. We find that for large neural networks an increase in damping both assists training and is even able to best the SGD test baseline. An interesting consequence of this finding is that it suggests hat damping should be considered an essential hyper-parameter in adaptive-gradient methods as it already is in stochastic second-order methods.

7 | Gadam/GadamX

In the last chapter, it was shown that Hessian based sharpness metrics actually increase when using $L2$ regularisation, which increase the generalisation performance. In this Chapter, we take this idea and see whether it is possible to construct an *Adaptive-gradient algorithm*, which leads to sharper optima and generalises better than finely tuned SGD. In order to lead the reader through some of the key content which led to the discovery of our adaptive algorithms with strong generalisation performance, we give an extensive overview, along with many theoretical results on *iterate averaging*. We note that many results are either special cases of more general results from the foundational work of Polyak and Juditsky [1992], with several of the results extended in Kushner and Yin [2003]. From the standpoint of readability, our results can be followed and understood by a much less mathematically literate audience. We posit that the background level of mathematics required to understand the results in [Polyak and Juditsky, 1992, Kushner and Yin, 2003] may have been a stumbling block in the wide-spread adoption of iterate averaging in deep learning. The Chapter structure will be as follows.

- We will introduce some key concepts from statistical learning theory, which allow us to give a precise definition of the word *generalisation*.
- We will give some background behind the optimisation literature and specifically methods of stochastic convex convergence proofs and their implications for algorithms in deep learning. We further analyse the effect of Polyak averaging on the noisy convex quadratic, compared to the final optimisation point and the exponentially weighted average. We prove that in the high-dimensional limit that reduction in variance

is more important than convergence in mean.

- we Introduce two novel algorithms *Gadam* and *GadamX* which combine Adam/Padam with Polyak averaging and weight decay which generalise significantly better than Adam/Padam and show results on a variety of architectures and datasets
- We show that our adaptive-gradient algorithms converge to much spectraly sharper solutions with higher generalisation for a combination of data-sets and network architectures.

This chapter content is centred around the submission *Gadam: Combining Adaptivity with Iterate Averaging Gives Greater Generalisation* which was a combined piece of work with Xingchen Wan. My personal contributions were all of the theoretical framework, including all the proofs about the noisy quadratic, high-dimensional geometry, weight reduction and the combination of Polyak averaging with the Adam algorithm. The insight that adaptive optimisers known to find sharper solutions could be adapted to give generalisation performance beyond that of flatter minima associated with SGD, inspired by the previous chapter, was also my own. Xingchen adapted the Gadam algorithm to include weight decay instead of $L2$ regularisation. He also developed GadamX, by combining Padam with Polyak averaging and decoupled weight decay. He is responsible took for most of the experiments, excepting the downsampled ImageNet WideResNet experiments.

7.1 Statistical Learning Theory

Formally, for some given prediction function $h(\cdot;\cdot) : \mathbb{R}^{d_x} \times \mathbb{R}^P \to \mathbb{R}^{d_y}$, we consider the family of prediction functions

$$\mathcal{H} := \{h(\cdot;\boldsymbol{w}) : \boldsymbol{w} \in \mathbb{R}^N\} \tag{7.1}$$

We aim to find the prediction function in this family that minimizes the losses incurred from inaccurate predictions. We assume a given loss function $l : \mathbb{R}^{d_{\boldsymbol{x}}} \times \mathbb{R}^{d_{\boldsymbol{y}}} \to \mathbb{R}$, yielding loss $l(h(\boldsymbol{x}_i;\boldsymbol{w})y)$ when $h(\boldsymbol{x}_i;\boldsymbol{w})$ and $\boldsymbol{y}_i$ are the respective predicted and true outputs. $d_{\boldsymbol{x}}, d_{\boldsymbol{y}}$ are the dimensionalities of the input and output respectively. Ideally we want to vary w such that we minimize the loss over our data generating distribution $P(x, y)$, this is known as the true or expected risk

$$R_{true}(\boldsymbol{w}) = \int_{\mathbb{R}^{d_{\boldsymbol{x}}} \times \mathbb{R}^{d_{\boldsymbol{y}}}} l(h(\boldsymbol{x};\boldsymbol{w}), y) dP(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}[l(h(\boldsymbol{x};\boldsymbol{w}), \boldsymbol{y})] \tag{7.2}$$

with corresponding gradient $g_{true} = \nabla R_{true}$ and Hessian $H_{true} = \nabla^2 R_{true}$. When we hold out a certain portion of the data and do not train the algorithm on it, denoted the *test set*, we can consider the risk on the test set as an unbiased estimate of the true risk. For a data-set with N training input-output samples, the empirical risk R_{emp}, is given by:

$$R_{emp}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} l(h(\boldsymbol{x}_i;\boldsymbol{w}), \boldsymbol{y}_i) \tag{7.3}$$

The minimisation of (7.3) is considered to be the practical optimisation problem of network training [Bottou et al., 2018]. Solutions which give high values of $R_{true}(\boldsymbol{w})$ and low values of $R_{emp}(\boldsymbol{w})$ are said to generalise poorly and the

difference between the two is known as the *generalisation gap*.

Overfitting: For a sufficiently flexible model parametrised by a large parameter vector $w \in \mathbb{R}^N$, for a loss such that $l(h(x_i; w), y_i) \geq 0 \, \forall w, x_i, y_i$, it may be possible to achieve $R_{emp}(w) \approx 0$ whilst $R_{true} \gg 0$, this is known as "over-fitting" the dataset. This is formalised in the field of learning theory using the concept of the Vapnik-Chervonenkis (VC) dimension of $\mathcal{H}$, denoted as $d_{\mathcal{H}}$. One of the most important results in learning theory, is that with probability of at least $1 - \eta$ the difference in true and empirical risk is less than

$$\sup_{h \in \mathcal{H}} |R_{true}(h) - R_{emp}(h)| \leq \mathcal{O}\left(\sqrt{\frac{1}{2T} \log \frac{2}{\eta}} + \frac{d_{\mathcal{H}}}{T} \log \left[\frac{T}{d_{\mathcal{H}}}\right]\right) \tag{7.4}$$

This bound blows up for large $d_{\mathcal{H}}$. The practical purpose of learning machines is not to learn the training set but to perform optimally on new unseen data. Hence Equation 7.3 is only important in so far as it faithfully represents Equation 7.2.

7.1.1 Cross-Entropy as a surrogate error

It is worth considering that in many problems of interest such as image classification, we may be interested in the proption of misclassifications, known as the error. This loss is formally defined as the indicator over the prediction function and the output.

$$E_{emp}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} 1[\hbar(\boldsymbol{x}_i; \boldsymbol{w}) \neq \boldsymbol{y}_i], \; 1[A] = \left\{ \begin{array}{ll} 1, & \text{if } A \text{ is true} \\ 0, & \text{otherwise} \end{array} \right\} \tag{7.5}$$

$$E_{true}(\boldsymbol{w}) = \int_{R^{d_{\boldsymbol{x}}} \times \mathbb{R}^{d_{\boldsymbol{y}}}} 1[\hbar(\boldsymbol{x}; \boldsymbol{w}) \neq y] dP(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}(1[\hbar(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y}]) \tag{7.6}$$

Where we use $\hbar(\boldsymbol{x}_i; \boldsymbol{w}) = \arg\max h(\boldsymbol{x}_i; \boldsymbol{w})$ to denote the most probable class given by the model. The held out test set error is an unbiased estimate of the true error. Whilst the training and test error during the optimisation trajectory are regularly reported, the non differentiability of the indicator function, means such a cost function is not amenable to gradient based methods. Usually a surrogate loss function is used, such as the cross-entropy loss defined as

$$l(h(\boldsymbol{x}_i; \boldsymbol{w}), \boldsymbol{y}_i) = -\sum_{c}^{d_{\boldsymbol{y}}} (1 - 1[\hbar(\boldsymbol{x}_i; \boldsymbol{w})_c \neq y_c]) h(\boldsymbol{x}_i; \boldsymbol{w})_c \tag{7.7}$$

The prediction function h may be a complicated function of the input $\boldsymbol{x}_i$, but very often in deep learning and the regime with which we exclusively concern ourselves in this thesis, the final layer is given by a softmax output $\sigma(\boldsymbol{z})$ of the input to the final layer $\boldsymbol{z}$

$$h(\boldsymbol{x}_i, \boldsymbol{w})_c = \sigma(\boldsymbol{z})_c = \frac{\exp \boldsymbol{z}_c}{\sum_{q=1}^{d_y} \exp \boldsymbol{z}_q} \tag{7.8}$$

What is immediately apparent from the combination of Equation 7.7 and Equation 7.8, is that even with zero error, the cross-entropy loss will not be zero, unless the output for the softmax output for the correct class is 1, which can only happen if $\boldsymbol{z}_j \to \infty$, hence the combination of cross-entropy loss and softmax not only penalises incorrect output but also the confidence of the output. The use of the cross-entropy (over another surrogate) can be inferred from its relationship to the relative entropy [Cover and Thomas, 2012], whose unique properties we discuss in Appendix A. As discussed in Chapter 4 the local error surface of both the error and cross-entropy loss is similar and hence captures the metric of interest.

7.2 Optimisation

For this Section we keep with the deep learning literature and denote the risk as defined in Section 7.1 as the *loss*, which is not to be confused with the loss function. In optimisation it is also common to not reduce the empirical risk Equation 7.3, but to use a different sub-sample of the dataset $B \ll N$ at each iteration and reduce what we denote as the batch risk

$$R_{batch}(\boldsymbol{w}) = \frac{1}{B}\sum_{i=1}^{B} l(h(\boldsymbol{x}_i; \boldsymbol{w}), \boldsymbol{y}_i) \tag{7.9}$$

7.2.1 SGD

With $L = R_{batch}$ in (Equation 7.9) as the the k-th iteration of SGD is of the form:

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha \nabla L(\boldsymbol{w}_k) \tag{7.10}$$

where α is the learning rate and $\nabla L(\boldsymbol{w})$ is the stochastic gradient of the loss L. augmenting SGD with momentum, we have

$$\begin{aligned} \boldsymbol{z}_{k+1} &\leftarrow \beta \boldsymbol{z}_k + \nabla L_k(\boldsymbol{w}) \\ \boldsymbol{w}_{k+1} &\leftarrow \boldsymbol{w}_k - \alpha \boldsymbol{z}_{k+1} \end{aligned} \tag{7.11}$$

where β, typically taken as 0.9.

7.2.1.1 Convergence of SGD on Convex problems

For β-Lipschitz, convex empirical risks, denoted the (overall) loss L. The difference between the $t+1$'th iterate and the optimal solution $L^*_{\boldsymbol{w}}$ can be bounded. The sum of differences along the trajectory (known as the *regret*)

telescopes, hence resulting in a convergence rate for the average regret which is an upper bound for the loss of the average point [Nesterov, 2013, Duchi, 2018]:

$$\begin{aligned}\delta L = L_{\boldsymbol{w}_{t+1}} - L_{\boldsymbol{w}^*} &\leq \nabla L_{\boldsymbol{w}_t}(\boldsymbol{w}_{t+1} - \boldsymbol{w}^*) + \frac{\beta}{2}||\boldsymbol{w}_{t+1} - \boldsymbol{w}_t||^2 \\ \mathbb{E}(\delta L) &\leq \hat{\nabla} L_{\boldsymbol{w}_t}(\boldsymbol{w}_t - \boldsymbol{w}^*) - (\alpha - \frac{\beta\alpha^2}{2})||\hat{\nabla} L_{\boldsymbol{w}_t}||^2 + \alpha\sigma_t^2\end{aligned} \tag{7.12}$$

where $\hat{\nabla} L_{\boldsymbol{w}_t}$ is the noisy gradient at $\boldsymbol{w}_t$ and σ_t^2 is its variance: $\text{Var}(\hat{\nabla} L_{\boldsymbol{w}_t})$. Noting that $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha\hat{\nabla} L_{\boldsymbol{w}_t}$:

$$\frac{R}{T} = \mathbb{E}\Big[\frac{1}{T}\sum_{t=1}^{T-1} L_{\boldsymbol{w}_{t+1}} - L_{\boldsymbol{w}^*}\Big] \tag{7.13}$$

Using Jensen's inequality, we have:

$$\begin{aligned}\frac{R}{T} &\leq \frac{1}{T}\sum_{t=0}^{T-1} \frac{||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 - ||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2}{2\alpha} + \alpha\sigma_t^2 \\ \mathbb{E}[L_{\frac{1}{T}\sum_{t=1}^{T-1}\boldsymbol{w}_{t+1} - \boldsymbol{w}^*}] &\leq \frac{R}{T} \leq \frac{||\boldsymbol{w}_0 - \boldsymbol{w}^*||^2}{2\alpha T} + \alpha\sigma_m^2\end{aligned} \tag{7.14}$$

where $\frac{1}{T}\sum_{t=1}^{T-1}\boldsymbol{w}_{t+1} - \boldsymbol{w}^*$ is known as the *Iterate Average* (IA) or *Polyak Average*, $\sigma_m^2 = \arg\max_{\boldsymbol{w}_t} \mathbb{E}||\hat{\nabla} L_{\boldsymbol{w}_t} - \nabla L_{\boldsymbol{w}_t}||^2$, and R is the regret. Setting $\alpha = (\beta + \sigma\frac{\sqrt{T}}{D})^{-1}$ in Equation 7.13 gives us the optimal convergence rate. Similar convergence results can be given for a decreasing step size $\alpha_t \propto t^{-1/2}\alpha_0$. Hence existing convergence results in the literature prove convergence for the iterate average, but not the final iterate.

7.2.1.2 Optimal Learning Rates

Setting $\alpha = (\beta + \sigma\frac{\sqrt{T}}{D})^{-1}$ gives us the optimal convergence rate of $\frac{\beta R^2}{T} + \frac{\sigma D}{\sqrt{T}}$. Similar convergence results can be given for a decreasing step size $\alpha_t \propto$

$t^{-1/2}\alpha_0$ [Duchi, 2018] when the number of iterations T is not known in advance. Given the use of both iterate averaging and learning rate schedule in the proofs, it is difficult to understand the relative importance of the two and how this compares with the typical heuristic of using the final point, which is common practice in most deep learning papers.

Drawback of second-order Optimizers: One of the drawbacks of second-order optimisers, is the cost of inverting a $\boldsymbol{H} \in \mathbb{R}^{P\times P}$ matrix which is an $\mathcal{O}(P^3)$ operation. This can be alleviated by using an iterative method and only looking at m approximate eigenvalue/eigenvector pairs, by using fast vector matrix multiplication which has a cost of $\mathcal{O}(2mPB)$. This is still $2m$ where a reasonable choice of $m \approx 30$ is often employed and hence the per iteration cost of iterative second-order methods is far greater than pure gradient methods and the benefits rarely outweigh the cost. The optimisation community has long considered gradient methods to be more stable to noise (due to sub-sampling) than second-order methods [Nesterov, 2013]. Martens and Grosse [2015] have developed heuristic second-order methods, which irregularly update the approximate curvature matrix to reduce computational cost, but these methods beyond having no guarantees of convergence are not regularly adopted. It is worth noting that the iterate average has a convergence rate which is similar to that of second-order methods, without the extra computation [Nesterov, 2013, Martens, 2014]

7.2.2 Adaptive-Gradient methods

Another strategy is to glean information about the curvature from the trajectory of the stochastic gradient iterates, but by approximating the inverse hessian term using only gradient information. Similar to second order opti-

misers, the iteration is generally in the form of:

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha \bar{\boldsymbol{H}}^{-1} \nabla L_k(\boldsymbol{w}) \tag{7.15}$$

where $\bar{\boldsymbol{H}}^{-1}$ is a pre-conditioning matrix capturing the curvature information. This is typically, approximated by heuristics such as the square-root of the second non central moment of the per parameter gradient. Similarly it may also contain momentum. To better understand the origins of this, we can consider the Regret bound R, of such an algorithm [Duchi et al., 2011]

$$R = \sum_{t=1}^{T} L(\boldsymbol{w}_t) - L(\boldsymbol{w}^*) \leq \frac{1}{2\alpha} ||\boldsymbol{w}_1 - \boldsymbol{w}^*||^2 + \frac{\alpha}{2} \sum_{t=1}^{T} \nabla L(\boldsymbol{w}_t) \bar{\boldsymbol{H}}^{-1} L(\boldsymbol{w}_t)^\dagger \tag{7.16}$$

The † denotes the conjugate transpose, which is equivalent to the transpose for real matrices and used to avoid confusing with the number of iterations T. Assuming that $\bar{\boldsymbol{H}}$ is full rank, we construct a Lagrangian for the problem $\mathcal{L} = R + \lambda(\text{Tr}\boldsymbol{H} \leq C)$ and minimising this Lagrangian and nothing that the second term in Equation 7.16 can be written as $\text{Tr}(\sum_{t=1}^{T} \nabla L(\boldsymbol{w}_t) \nabla L(\boldsymbol{w}_t)^\dagger \bar{\boldsymbol{H}}^{-1}))$ we have that

$$\begin{aligned} \delta\mathcal{L} = 0 = \text{Tr}(-\bar{\boldsymbol{H}}^{-2} \sum_{t=1}^{T} \nabla L(\boldsymbol{w}_t) \nabla L(\boldsymbol{w}_t)^\dagger) + \text{Tr}\lambda \boldsymbol{I} \\ \therefore \bar{\boldsymbol{H}} \propto \left(\sum_{t=1}^{T} \nabla L(\boldsymbol{w}_t) \nabla L(\boldsymbol{w}_t)^\dagger \right)^{-1/2} \end{aligned} \tag{7.17}$$

The proof for the general case can be found in [Duchi et al., 2011]. Hence Adam [Kingma and Ba, 2014] can be seen as an exponentially weighted diagonal approximation of Equation 7.17. Martens [2014] argues that this can be seen as simply trusting the curvature information less. The use of a power less than 1/2, such as 1/8 by Chen and Gu [2018], which we adopt as a ref-

erence in our later experiments can be seen as further limiting the trust of these adaptive methods.

7.2.2.1 Convergence of Adaptive methods

For adaptive optimisers, the noisy gradient is preconditioned by some non-identity matrix $\bar{\boldsymbol{B}}^{-1}$:

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - \alpha \bar{\boldsymbol{H}}^{-1} \nabla L_k(\boldsymbol{w}) \tag{7.18}$$

Methods of proof [Reddi et al., 2018, Tran et al., 2019] rely on bounding the regret $\mathcal{O}(\sqrt{T})$ and showing that the average regret $\frac{R}{T} \rightarrow 0$ and Equation 7.13 explicitly demonstrates that the average regret is an upper bound on the expected loss for the average point in the trajectory. These methods are typically much cheaper than second-order methods.

7.3 Gadam/GadamX

Here we present the full Gadam/GadamX algorithm. Note that for simplicity, in Algorthm 9 we present a Polyak-style averaging of every iteration, in practice we find both practical and theoretical results why averaging *less* frequently is almost equally good, if not better. We include a discussion on this in Section 7.8.6. The difference between Gadam and GadamX is that for the latter we use partially adaptive Adam [Chen and Gu, 2018]. This changes the square-root of the moving average of the second momemnt of the parameters in the denominator to some other (smaller) power, to make the update more SGD like. Zhou et al. [2018] argue that for non convex optimisation, the convergence of partially adaptive Adam is superior to AMSgrad (the theoretically justified version of Adam [Reddi et al., 2018]).

Algorithm 9 Gadam/GadamX

Require: initial weights θ_0; learning rate scheduler $\alpha_t = \alpha(t)$; momentum parameters $\{\beta_1, \beta_2\}$ (Default to $\{0.9, 0.999\}$ respectively); partially adaptive parameter $p \in [0, 0.5]$ Default to $\{0.125, 0.5\}$ for {GadamX, Gadam}; decoupled weight decay λ; averaging starting point T_{avg}; tolerance ϵ (default to 10^{-8})

Ensure: Optimised weights $\tilde{\theta}$

Set $\boldsymbol{m}_0 = 0, \boldsymbol{v}_0 = 0, \hat{\boldsymbol{v}}_0 = 0, n_{\text{models}} = 0$.

for t = 1, ... T **do**

$\alpha_t = \alpha(t)$

$\boldsymbol{g}_t = \nabla f_t(\theta_t)$

$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t / (1 - \beta_1^t)$

$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2 / (1 - \beta_2^t)$

$\hat{\boldsymbol{v}}_t = \max(\hat{\boldsymbol{v}}_{t-1}, \hat{\boldsymbol{v}}_t)$ (If using Amsgrad)

$\theta_t = (1 - \alpha_t \lambda)\theta_{t-1} - \alpha_t \frac{\hat{\boldsymbol{m}}_t}{(\hat{\boldsymbol{v}}_t + \epsilon)^p}$

if $T \geq T_{\text{avg}}$ **then**

$n_{\text{models}} = n_{\text{models}} + 1$

$\theta_{\text{avg}} = \frac{\theta_{\text{avg}} \cdot n_{\text{models}} + \theta_t}{n_{\text{models}} + 1}$

else

$\theta_{\text{avg}} = \theta_t$

end if

end for

return $\tilde{\theta} = \theta_{\text{avg}}$

7.3.1 Drawbacks of Gadam

One clear drawback of Adam is that we need to hold another extra vector in memory, whilst theoretically this could be done on the CPU, independently from the gradient trajectory on the GPU, this is not the case in our implementation. We note that momentum requires an extra vector of parameter length and this is similar (but is not called as regularly as momentum so could be much cheaper if implemented as such). A much more costly operation which is not detailed in the Algorithm is that we must compute the batch normalisation statistics for the IA point (this cannot be averaged and

attempting to do so naively leads to very poor results). In our implementation this requires an entire forward pass of the data. For small datasets like CIFAR-10/100 this is not an appreciable concern. However for ImageNet with over 1 millions images, this is appreciable. Note such a pass of a data essentially doubles the cost for each epoch which utilises IA. In order to remedy this we suggest two solutions

- Do not compute the forward pass on the IA until the very last epoch (which is always the best in our experiments)
- Use a sub-sample of the training data to compute the batch normalisation statistics and reduce the computation

which could be implemented as future work. The reason we do not implement the first suggested item is because we want to show the improvement in each epoch of IA. For a practitioner this is may not be necessary and so the cost of IA can reduce to a single extra epoch if the full training set is used for the forward pass and less if not. We comment that in our work we do not find that more IA epochs lead to overfitting.

7.4 Validating Experiments

It is curious why popular adaptive methods (such as Adam) are not commonly used in combination with IA, despite the desirable theoretical properties of IA and the fact that authors of Adam even suggest Polyak-style IA as a possible enhancement [Kingma and Ba, 2014]. While we believe this can be partly attributed to the fact that IA is often seen as a theoretical tool that is sometimes labelled as not useful practically [Trivedi and Kondor, 2017], we also find naively combining Adam with IA (denoted as *Adam-IA* hereafter)

to be uncompetitive, at least in the vision tasks. With reference to Figure 7.1, even though the iterates of Adam (solid lines) perform better than SGD (dotted lines) when we start averaging, the improvement from averaging and the final accuracy is much worse in Adam-IA. However, rather than attributing for this weak performance to the adaptive nature of Adam itself, we argue that the real culprit is the ineffective regularisation. Indeed, by comparing the weight norm of Adam and AdamW [Loshchilov and Hutter, 2019] in Figure 7.1(b), it is apparent that AdamW exerts a much stronger weight reduction effect. Weight reduction, as shown in Section 7.5.6 is *not* peculiar to IA; simply decaying the learning rate can already lead to a comparable, if not larger, amount of weight reduction. However, modern CNNs are almost universally equipped with normalisation schemes such as BN. Weight reduction achieved through L_2 regularisation, which is staple of DNN training, works *not* in the classical way in limiting the expressiveness but rather by encouraging high effective learning rate that scales $\alpha_{\text{eff}} \propto \frac{\alpha}{||\boldsymbol{w}||^2}$[Zhang et al., 2018]. We note that there has been extensive work empirically demonstrating that large learning rates lead to greater generalisation [Jastrzkbski et al., 2017].

7.4.1 Validating our Assertions

We argue that the real value in combining Adam with IA and decoupled weight decay is the ability to converge with a high effective learning rate. This implies 1) we expect Gadam (and indeed all optimisers with IA, but we focus on Gadam here) to be more effective with BN since it both keeps α large and $||\boldsymbol{w}||^2$ small. Given that much criticism about adaptive methods is their propensity to "over-adapt" and the "small learning rate dilemma" (i.e., learning rate decaying combined with adaptive momentum leading to

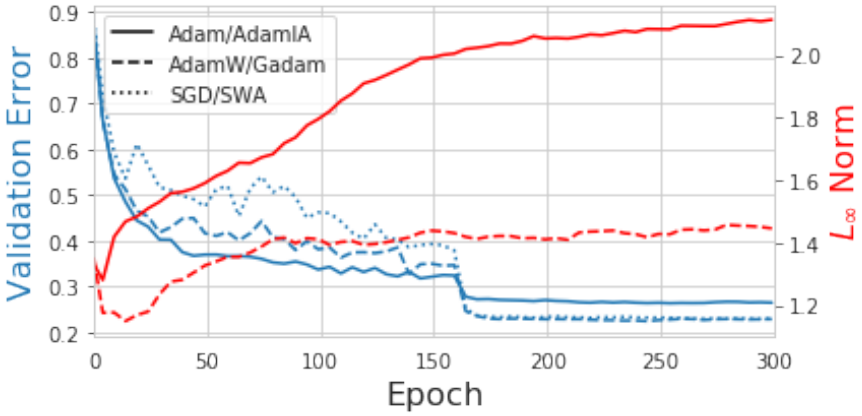
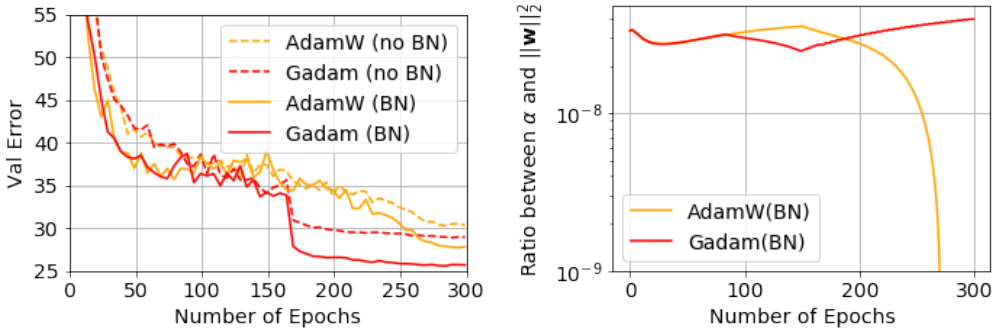


Figure 7.1: **Combining Adam with Iterate Averaging does not work because of ineffective regularisation, implementing decoupled weight decay fixes this.** Comparison of validation error and iterate L_∞ norm of Adam/Adam-IA and AdamW/Gadam on PRN-110 CIFAR-100.

no training progress made) [Chen and Gu, 2018, Wilson et al., 2017], this could provide much-needed regularisation and escape from the dilemma; 2) explicitly reducing α has little regularisation benefit, as the aggressive decay easily eclipses any weight reduction. We experimentally validate both are true using a classical VGG-16 network on CIFAR-100 with and without BN (Fig. 7.2(a)(b)): under the identical setup (detailed in Section 7.4), the margin of improvement of Gadam is much larger with BN. While Gadam keeps $\frac{\alpha}{||\boldsymbol{w}||^2}$ high, in scheduled AdamW it quickly vanishes once we start learning rate decay. It is then curious to ask whether using AdamW with no/more moderate schedule can counteract this. However, we find either to underperform the scheduled AdamW by more than 5% in accuracy rendering their discussions irrelevant (not shown in the graph).

Why the VGG-16? In this expository experiment, we use the original VGG-16. We select VGG-16 as it is both a very classical network and is also rather unique in a sense that its variants both with and without batch normalisation [Ioffe and Szegedy, 2015] (BN) are widely used. More recent networks



(a) Val Error (b) $\alpha/||\boldsymbol{w}||^2$

Figure 7.2: Val. error and $\frac{\alpha}{||\boldsymbol{w}||^2}$ of CIFAR-100 on VGG-16 with and without BN. In (b) only results with BN where the quantity is relevant are shown.

such as ResNets almost invariably have BN built-in, and their counterparts without BN are not commonly used. Experiments that train ResNets without batch normalisation Ghorbani et al. [2019] use learning rates hundreds of times smaller than those typical with batch-normalisation and achieve very poor performance. We conduct all experiments with initial learning rate 0.03. For fair comparison to previous literature, we use the linear decay schedules advocated in [Izmailov et al., 2018], for both SGD and IA. For IA we run the set of terminal learning rates during averaging $\{0.03, 0.01, 0.003\}$, whereas for SGD we decay it linearly to 0.0003.

7.5 Theoretical Exposé: Noisy Quadratic

One of the key adaptations we make to the Adam algorithm is to replace typically sharply decaying learning rate schedules with Polyak averaging, i.e. iterate averaging (IA). As stressed in the introduction, there is extensive theoretical justification for this already in the literature [Polyak and Juditsky, 1992, Kushner and Yin, 2003], however in practice most practitioners do not employ Polyak averaging and instead decay the learning rate, returning

the final iterate as the solution. Exponentially weighted/moving averages (EWA/EMA) are also used [Martens, 2014].

In order to compare such methods against IA, we consider the $1D$ quadratic function $f(w) = \frac{\lambda}{2}||w||^2$ as a proxy of our true loss function, minimised using gradient descent with learning rate α. At each iteration the gradient is perturbed by some i.i.d. Gaussian noise, $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The final iterate and iterate average after a total training budget n are given by:

$$w_n \sim \mathcal{N}\left((1-\alpha\lambda)^n w_0, \frac{\alpha\sigma^2(1-(1-\alpha\lambda)^{n-2})}{\lambda}\right), w_{\text{avg}} \sim \mathcal{N}\left(\frac{[1-(1-\alpha\lambda)^{n-2}]w_0}{n\alpha\lambda}, \frac{\alpha\sigma^2}{\lambda n}\right)$$

where we have exploited the formulas for geometric sums and independence and Gaussianity of the noise and upper bounded the variance of the iterate average. Let us now assume that $n \to \infty$ and that $|\alpha\lambda| \ll 1$ Then the terms above simplify to

$$w_n \sim \mathcal{N}\left(w_0 \mathrm{e}^{-n\alpha\lambda}, \frac{\alpha\sigma^2}{\lambda}\right), \; w_{\text{avg}} \sim \mathcal{N}\left(\frac{w_0}{n\alpha\lambda}, \frac{\alpha\sigma^2}{\lambda n}\right)$$

Whilst we attain exponential convergence in the mean for the end point, we do not control the noise in the gradient, whereas for the averaged iterates, although the convergence in the mean is worse (linear), the variance vanishes asymptotically. In higher dimensions, where P is large, the iterates evolve independently along the eigenbasis of $\boldsymbol{H} = \nabla^2 L$ and assuming isotropic noise:

$$\boldsymbol{w}_n \sim \mathcal{N}\left(\sum_{j=1}^{P} w_{0,j}\mathrm{e}^{-n\alpha\lambda_j}, \sum_{j=1}^{P}\frac{\alpha\sigma^2}{B\lambda_j}\right), \boldsymbol{w}_{\text{avg}} \sim \mathcal{N}\left(\sum_{j=1}^{P}\frac{w_{0,j}}{n\alpha\lambda_j}, \sum_{j=1}^{P}\frac{\alpha\sigma^2}{B\lambda_j n}\right)$$

where B is the minibatch size. In the asymptotic limit $n \to \infty$, $\boldsymbol{w}_{\text{avg}}$ converges to the minimum, whereas the $\boldsymbol{w}_n$ does not. In high dimensions, the low dimensional intuition that majority of the probability mass is concentrated

around the mean fails, as the relevant quantity is not the probability density at a point, but the integral under the density in the immediate vicinity of that point, which scales as $p(r)r^{P-1}dr$ for P dimensions. Formally,

Theorem 18. *Under the assumptions in the preceding Section*

$$\begin{aligned}\mathbb{P}\left\{||\boldsymbol{w}_n|| - \sqrt{\sum_i^P w_{0,i}^2 \mathrm{e}^{-2n\alpha\lambda_i} + P\frac{\alpha\sigma^2}{B}\langle\frac{1}{\lambda}\rangle} \geq t\right\} \leq \nu \\ \mathbb{P}\left\{||\boldsymbol{w}_{\text{avg}}|| - \sqrt{\sum_i^P \frac{w_{0,i}^2}{\lambda_i^2 n^2\alpha^2} + \frac{P\alpha\sigma^2}{Bn}\langle\frac{1}{\lambda}\rangle} \geq t\right\} \leq \nu\end{aligned} \tag{7.19}$$

where $\nu = 2\exp(-ct^2)$ *and* $\langle\lambda^k\rangle = \frac{1}{P}\mathrm{Tr}\boldsymbol{H}^k$

7.5.1 Proof of Theorem 18

The basic idea, is that since

$$\mathbb{E}||\boldsymbol{w}||_2^2 = \sum_i^p \mathbb{E}(w_i^2) = \sum_i^p \Big((\mathbb{E}(w_i))^2 + \mathbb{V}(w_i)\Big) \tag{7.20}$$

we expect $||\boldsymbol{w}||_2$ to be approximately the square-root of this. Our proof follows very closely from Vershynin [2018] (p.51) except that the variables we consider are not zero-mean or unit-variance. We sketch the proof below:

Proof Sketch: To show that Theorem 18 is indeed true with high probability, we consider the centred, unit variance version of the random variables i.e. $X_i = (\tilde{X}_i - \mu_i)/\sigma_i$

Lemma 1 *(Bernstein's inequality)*: Let $\{X_1, ..X_n\}$ be independent, zero mean, sub-exponential random variables. Then for every $t \geq 0$, we have

$$\mathbb{P}\left\{\left|\frac{1}{N}\sum_i\sum_i^N X_i\right| \geq t\right\} \leq 2\exp\left\{-c\min\left(\frac{t^2}{K^2}, \frac{t}{K}\right)N\right\} \tag{7.21}$$

where $K = \arg\max_i ||X_i||_{\phi_1}$, and $||X||_{\phi_i} = \inf\{t > 0 : \mathbb{E}\exp|X|/t \leq 2$.

Proof. The proof is standard and can be found in Vershynin [2018] p.45, essentially we multiply both sides of the inequality by λ, exponentiate, use Markov's inequality and independence assumption. Then we bound the moment generating function of each X_i and optimise for λ □

Let $X = (X_1, ..., X_n) \in \mathbb{R}^P$ be a random vector with independent sub-Gaussian coordinates X_i, that satisfy $\mathbb{E}X_i^2 = 1$. We then apply Bernstein's deviation inequality (Lemma 1) for the normalised sum of independent, mean zero variables

$$\frac{1}{P}||X||_2^2 - 1 = \frac{1}{n}\sum_i^P (X_i^2 - 1) \tag{7.22}$$

Since X_i is sub-Gaussian $X_i^2 - 1$ is sub-exponential and by centering and the boundedness of the MGF $||X_i^2 - 1||_{\phi_1} \leq CK^2$ hence assuming $K \geq 1$

$$\mathbb{P}\left\{\left|\frac{1}{P}||X||_2^2 - 1\right| \geq u\right\} \leq 2\exp\left(-\frac{cP}{K^4}\min(u^2, u)\right) \tag{7.23}$$

Then using $|z - 1| \geq \delta$ implies $|z^2 - 1| \geq \max(\delta, \delta^2)$

$$\mathbb{P}\{\left|\frac{1}{\sqrt{P}}||X||_2 - 1\right| \geq \delta\} \leq \mathbb{P}\{\left|\frac{1}{P}||X||_2^2 - 1\right| \geq \max(\delta, \delta^2)\} \leq 2\exp -\frac{cP}{K^4\delta^2} \tag{7.24}$$

changing variables to $t = \delta\sqrt{P}$ we obtain

$$\mathbb{P}\{||X||_2 - \sqrt{P}| \geq t\} \leq 2\exp -\frac{ct^2}{K^4} \tag{7.25}$$

for all $t \geq 0$ Our proof follows by noting that the significance of the 1 in Equation 7.24 is simply the mean of the square and hence by replacing it by the mean squared plus variance we obtain Theorem 18. ■

Implications of Theorem 18

In the high-dimensional regime typical of deep learning, the effect of noise may very well dominate the convergence in the mean, *provided one starts averaging in a region reasonably close to the minimum* - this highlights why one should only start averaging when a specific metric (e.g. validation accuracy) stagnates. While the iterates will be located in a thin-shell with a high probability, the robustness to the gradient noise will drive the IA closer to the minimum. Unless the per parameter estimation noise scales $\propto \frac{1}{P}$ which is an odd assumption to make, we expect *this effect to be more and more significant when the number of parameters P gets larger*. With P being a rough gauge of the model complexity, this implies that in more complex, over-parameterised models, we expect the benefit of IA to be larger. Theorem 18 also unifies the various scheduling approaches and highlights why IA is a desirable alternative. Observing the σ^2 dependence in Theorem 18, reducing the learning rate α is an obvious way to reduce the noise effect. However, there is a limit to how much one may reduce α: for very small α, $\boldsymbol{w}_n$ converges at a rate $(1 - n\alpha\lambda)$ whereas $\boldsymbol{w}_{\text{avg}}$ converges at $(1 - \frac{n\alpha\lambda}{2})$. An alternative is increasing the batch size B. However, the maximal batch size possible is using the full data-set, which for deep learning is significantly smaller than the total num-

ber of parameters $N \ll P$. Furthermore, the number of passes through the data is typically kept fixed, so $n = \frac{NE}{B}$, where E is the number of epochs. For the same computational budget, the convergence in the mean will also be reduced. Hence for a large number of functional evaluations, it can be seen that IA will outperform both reducing the learning rate and increasing the batch size.

7.5.2 Exponentially-weighted Moving Average (EMA)

Various previous works also use EMA in weight space to achieve optimisation and/or generalisation improvements: Izmailov et al. [2018] entertain the use of EMA in SWA, although they conclude simple averaging is more competitive. Very recently, Zhang et al. [2019b] proposes *Lookahead*, a plug-in optimiser that uses EMA on the slow weights to improve convergence and generalisation. Nonetheless, having argued the dominance of noise in the high-dimensional deep learning regime, we argue that simple averaging is more theoretically desirable *for generalisation*: following the identical analysis to Section 7.5, we keep in the $1D$ case w.l.o.g and denote $\rho \in [0, 1]$ as the coefficient of decay, asymptotically the EMA point $\boldsymbol{w}_{\text{ema}}$ is governed by the normal distribution:

$$\mathcal{N}\left(\frac{(1-\rho)w_0(1-\alpha\lambda)^{n+1}[1-(\frac{\rho}{1-\alpha\lambda})^{n-1}]}{1-\alpha\lambda-\rho}, \frac{1-\rho}{1+\rho}\frac{\alpha\sigma^2\kappa}{\lambda}\right)$$

where $\kappa = (1-(1-\alpha\lambda)^{n-2})$. An alternative analysis of EMA arriving at similar result was also done in Zhang et al. [2019a], but their emphasis of comparison is between the EMA point and *iterates* instead of the former and the *IA point* in our case. Therefore, although the convergence in mean is less strongly affected, the noise is reduced by a factor of $\frac{1-\rho}{1+\rho}$. So whilst we

reduce the noise (possibly by a very large factor), it does not vanish in the asymptotic limit. Hence viewing EMA or IA as noise reduction schemes, we can consider IA a far more aggressive noise reduction scheme than EMA. Secondly, EMA implicitly assumes that more recent iterates are better, or otherwise more important, than the previous iterates. While justified initially (partially explaining EMA's efficacy in accelerating optimisation), it is less so in the late stage of training when we typically start averaging. Indeed, we argue that in terms of asymptotic expected loss, IA will *always* better the final iterates with realistic schedules. Building on previous analysis of quadratics [Martens, 2014], we have:

Theorem 19. *Denote (the positive semidefinite surrogate of) Hessian as* $\boldsymbol{H}$ *and covariance matrix of gradients as* $\boldsymbol{\Sigma}_g$*, if we further assume* $\boldsymbol{H}$ *and* $\boldsymbol{\Sigma}_g$ *are co-diagonalisable [Zhang et al., 2019a], then for the expected loss of the final iterate to be smaller than that of the average iterate, the learning rate must decay in a rate* $\propto \frac{1}{n}$.

This implies a learning rate decay greater than the theoretically proposed schedule of $\alpha \propto \frac{1}{\sqrt{n}}$. This schedule is already almost never used in practice, as it decays the learning rate too fast and strongly harms the convergence in mean.

7.5.3 Proof of Theorem 19

Building on the quadratic analysis in Martens [2014], we first consider result for a fixed learning rate pure gradient method where, the final/average iterate will tends to a loss $\mathbb{E}L(\boldsymbol{w}_n)$, $\mathbb{E}L(\boldsymbol{w}_{avg,n})$ respectively as $n \to \infty$:

$$\begin{aligned}\mathbb{E}\Big(L(\boldsymbol{w}_n)\Big) &\leq L(\boldsymbol{w}^*) + \frac{\alpha}{4}\mathrm{Tr}\Big((1-\frac{\alpha}{2}\boldsymbol{H})^{-1}\boldsymbol{\Sigma}_g(\boldsymbol{w}^*)\Big) \\ \mathbb{E}\Big(L(\boldsymbol{w}_{avg,n})\Big) &\leq L(\boldsymbol{w}^*) + \frac{1}{2n+2}\mathrm{Tr}\Big(\boldsymbol{H}^{-1}\boldsymbol{\Sigma}_g(\boldsymbol{w}^*)\Big)\end{aligned} \tag{7.26}$$

where $\boldsymbol{\Sigma}_g$ is the gradient noise covariance. Thus, for the expected loss of the iterate to be lower, we need:

$$\frac{\alpha}{4}\mathrm{Tr}\Big((1-\frac{\alpha}{2}\boldsymbol{H})^{-1}\boldsymbol{\Sigma}_g(\boldsymbol{w}^*)\Big) \leq \frac{1}{2n+2}\mathrm{Tr}\Big(\boldsymbol{H}^{-1}\boldsymbol{\Sigma}_g(\boldsymbol{w}^*)\Big) \tag{7.27}$$

Simplifying yields

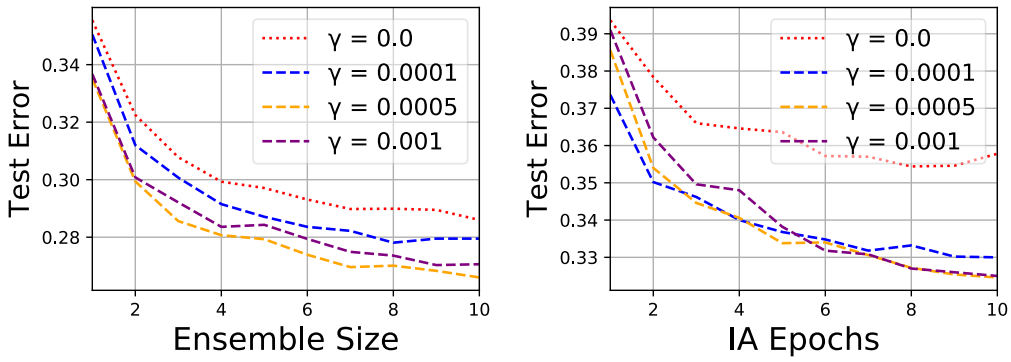
$$\alpha \leq \frac{2}{2+n}\frac{\mathrm{Tr}\boldsymbol{H}^{-1}}{P} \tag{7.28}$$

which requires $\alpha \propto \frac{1}{n}$. ■

Note on Assumptions: We make the similar assumption as Roux et al. [2008] that the batch gradients are draws from the *true* gradient distribution and all gradients are i.i.d. By the Central Limit Theorem, the analysis of Section 7.5 holds.

7.5.4 Comparison to Bayesian Model Averaging

Since a greater regularisation increases the model bias (towards zero weights), the decrease in mean square error (as measured by test set performance), can only be from a reduction in variance (since we have not changed the model or the dataset we do not alter the fundamental error). From a modelling ensembling perspective [Schapire, 1990], averaging only improves test performance for sufficiently uncorrelated models (through a reduction in variance). The effective weight decay per iteration is given by $(1-\alpha\gamma)$ and hence for the same learning rate, we expect schedules with larger regularisation co-



(a) Network Ensemble (b) Iterate Averaging

Figure 7.3: **Importance of Regularisation for IA.** Test error improvement as a function of regularisation γ for network ensembling and Iterate Averaging (IA) for the VGG-16 on the CIFAR-100 dataset for 100 epochs, learning rate $\alpha = 0.02$, decayed by 100 for SGD ensembles and decayed by a factor of 2 for the final IA epochs.

efficients γ to have a greater solution difference between epochs and hence benefit more from IA. Whilst Izmailov et al. [2018] argue for the importance of large learning rates in conjunction with IA to achieve strong generalisation performance, the effective weight decay, which controls the extent to which we "forget" the previous model, is a product of regularisation coefficient and learning rate. **This indicates that weight decay is essential for the practical performance improvements of IA**. We showcase this in Fig 7.3b, where no regularisation corresponds to the lowest improvement in test error as we employ IA. Furthermore, even as increasing the weight decay coefficient decreases test performance before starting IA, the improvement thereafter (due to the decreased correlation between the averaged models) more than makes up for this difference leading to better performance. We note that this is in contrast to traditional ensembling (where we average the predictions not the weights, Fig 7.3a), where the extent of improvement is largely agnostic to the weight decay co-efficient.

7.5.5 Implicit Learning Rate Decay in IA

To achieve the best performance, deep learning practitioners employ learning rate *scheduling*. While it is often argued that adaptive optimisers do not *need* scheduling to converge, scheduling improves their empirical performance [Loshchilov and Hutter, 2019, Wilson et al., 2017] and is required in proofs of convergence [Reddi et al., 2018]. For constant high learning rates, the gap between training and test curves is small, but the final test performance is poor. Whilst decaying the learning rate increases test performance, the generalisation gap also increases as shown in Figure 7.4, suggesting more over-fitting. This prompts us to consider whether we can preserve the regularisation benefits of large learning rates, whilst improving the final test performance.

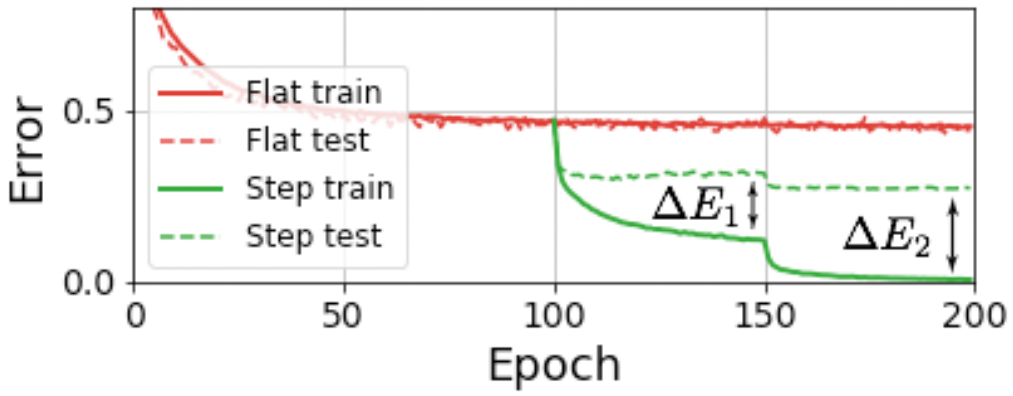


Figure 7.4: An example illustrating the *generalisation gap*. We run VGG-16 on CIFAR-100, with *flat* learning rate of 0.1, and with *step decay* with a factor of 10 at $\{100, 150\}$-th epochs. It can be seen that at high constant learning rate there is no generalisation gap, but as we decay more and more the test performance improves but the gap also increases ($\Delta E_2 > \Delta E_1$).

As we discussed in introduction, most recent works running IA with high constant learning rate, have mainly considered IA as a *generalisation* technique, and have not explored its connection to the more commonly employed

learning rate scheduling which we argue to exist. Denoting $\boldsymbol{w}_0$ as the point at which averaging starts and $\boldsymbol{g}_i$ the gradient at point i and assuming that learning rate α is kept constant for the duration of averaging (which is always the case in this Chapter), the average of the iterates is given by

$$\boldsymbol{w}_0 + \frac{n-1}{n}\alpha\boldsymbol{g}_0 + ... + \frac{n-k}{n}\alpha\boldsymbol{g}_{k-1}\,, 1 \leq k \leq n-1 \tag{7.29}$$

and hence we linearly reduce the effective learning rate. It is worth noting that when we begin averaging, we update both n and k on the fly. This means that for the first iterate of averaging, we have essentially halved the learning rate

$$\boldsymbol{w}_{avg,1} = \frac{\boldsymbol{w}_0 + \boldsymbol{w}_0 - \alpha g_0}{2} = \boldsymbol{w}_0 - \frac{1}{2}\alpha g_0 \tag{7.30}$$

Hence, we expect the training curves to more closely resemble that of step decay, which is what we observe in practice. However, the benefits of IA extend beyond implicit learning rate decay and we analyse this in the following Section.

7.5.6 Regularising Effects of IA

Related to our first claim, we also argue that IA exerts regularisation through weight reduction. Formally:

Theorem 20. *If iterates $\{\boldsymbol{w}_i\}$ are drawn from a uniform distribution in the weight space, then the L_2 norm of the expected IA point is smaller or equal to the expectation of the L_2 weight norm of the iterates.*

$$||\mathbb{E}(\boldsymbol{w}_i)||_2^2 \leq \mathbb{E}\bigg(||\boldsymbol{w}_i||_2^2\bigg) \tag{7.31}$$

Proof. For $T = 2$

$$\frac{(\boldsymbol{w}_1 + \boldsymbol{w}_2)^2}{4} \leq \frac{\boldsymbol{w}_1^2 + \boldsymbol{w}_2^2}{2} \tag{7.32}$$

$$0 \leq (\boldsymbol{w}_1 - \boldsymbol{w}_2)^2 \tag{7.33}$$

Assume for $T = n$, for $T = n + 1$ i.e. $(\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{w}_i)^2 \leq \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{w}_i^2$

$$\left(\frac{1}{n+1}\sum_{i=1}^{n+1} \boldsymbol{w}_i\right)^2 \leq \frac{1}{n}\sum_{i=1}^{n+1} \boldsymbol{w}_i^2 + \sum_{i=1}^{n} \boldsymbol{w}_i^2 \tag{7.34}$$

$$\left(\sum_{i=1}^{n} \boldsymbol{w}_i\right)^2 + 2\boldsymbol{w}_{n+1}\sum_{i=1}^{n} \boldsymbol{w}_i + \boldsymbol{w}_{n+1}^2 \leq n\sum_{i=1}^{n} \boldsymbol{w}_i^2 + \sum_{i=1}^{n} \boldsymbol{w}_i^2 + (n+1)\boldsymbol{w}_{n+1}^2 \tag{7.35}$$

$$0 \leq \sum_{i=1}^{n} (\boldsymbol{w}_i - \boldsymbol{w}_{n+1})^2 \tag{7.36}$$

and by mathematical induction, for any $T = n \geq 2$ and that $n \in \mathbb{N}^+$, we have:

$$\left(\frac{1}{T}\sum_{i=1}^{T} \boldsymbol{w}_i\right)^2 \leq \frac{1}{T}\sum_{i=1}^{T} \boldsymbol{w}_i^2 \tag{7.37}$$

Assuming that $\{\boldsymbol{w}_i\}$ are drawn from a uniform distribution, Theorem 20 follows. ■ □

The assumption on the uniform distribution is more likely to be met near the end of training when the validation statistics stop improving and iterates oscillate around the minimum. Hence, by Theorem 20, we expect the weight norm of the IA point to be less than the expected weight norm of the individual iterates. Low weight norm solutions correspond to maximum margin solutions, which have a fruitful history in machine learning and have been touted as a major reason for the implicit regularisation of non-adaptive methods [Wilson et al., 2017]. L_2 regularisation is extensively adopted and

can be shown to reduce the effect of static noise on the targets [Krogh and Hertz, 1992]. To showcase the relevance of Theorem 20, we plot the weight norm of the VGG-16 on the CIFAR-100 data-set along with test accuracy (Figure 7.5). We see that as soon as iterate averaging is enabled at epoch 161 that the weight norm begins to decrease from that of the iterates and that the accuracy of the iterate average surpasses that of the iterates. The extent of weight reduction will depend on the variance and independence in the weights of the iterates element-wise, and hence we only expect a significant reduction for large moves in the weight-space, *corresponding to a high learning rate* - this could explain why in IA, a high learning rate usually leads to the best performance.

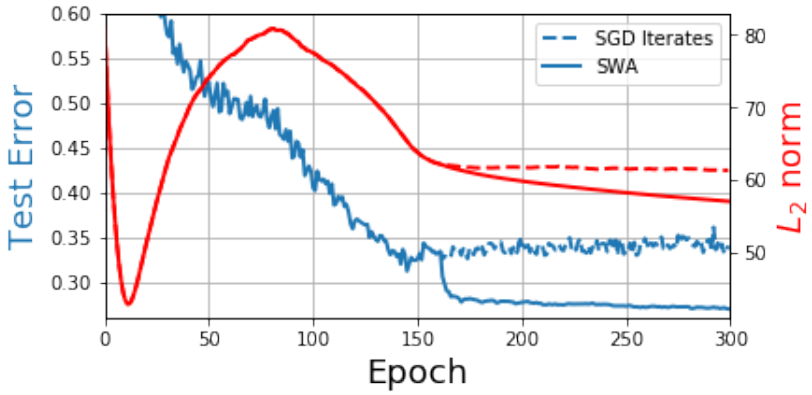
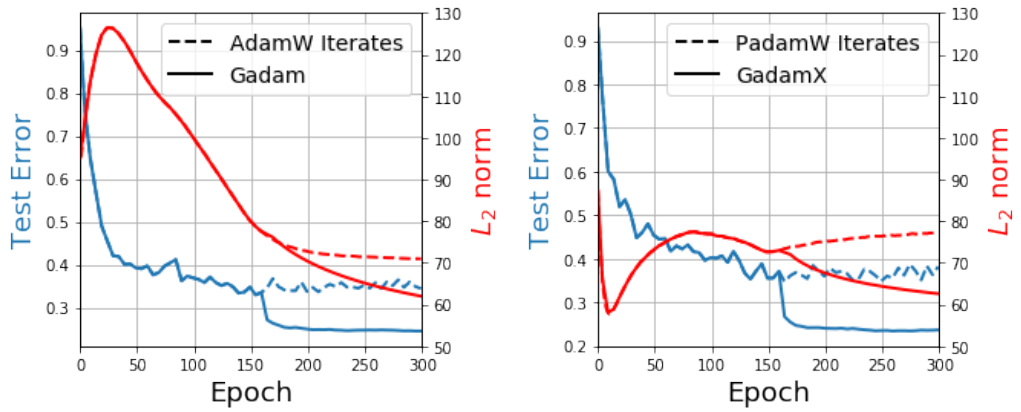


Figure 7.5: Test errors and L_2 weight norms of the IA point against the iterates of VGG-16 on the CIFAR-100 dataset. Initial learning rate 0.05 and the learning rate at the inception of IA is 0.01.

On the regularising effect: It is apparent that Theorem 20 is not specific on any particular choice of optimiser. Nonetheless, we empirically show it that in simple VGG-16 experiments on CIFAR-100 that the weight-reducing regularising effect is present in all optimisers with IA. The results are shown in Figure 7.6. It is also very interesting to see that although fully adaptive AdamW iterates initially lead to a very high L_2 weight norm, with effective

regularisation the optimiser nevertheless finds a L_2 solution comparable to SGD/SWA eventually.



(a) AdamW/Gadam (b) PadamW/GadamX

Figure 7.6: Regularising Effect of IA in different optimisers.

7.6 Applicability of Theoretical Analysis for Adaptive-Gradient Methods

We make the claim that despite of using pure, non-adaptive-gradient descent as the analysis tool for derivations of our theoretical results, the results apply in adaptive optimisation. Here we substantiate that claim by establishing the applicability of our arguments in Section 7.5.5 to adaptive optimisation, both theoretically and experimentally.

On expected loss from averaging: On our analysis on the quadratic optimisation with noisy gradients, we first argue that *with IA there is an asymptotic equivalence between SGD and adaptive methods*, and therefore all our theoretical results in Section 7.5 hold. In Martens [2014], the authors derive general bounds on the loss of a noisy quadratic and show that, for

methods that precondition the gradient with a non-identity matrix $\boldsymbol{B}^{-1}$ (including adaptive methods such as Adam), for a fixed step size α the n-th iterate and the average of the iterates will tend to losses $\mathbb{E}L(\boldsymbol{w}_n)$, $\mathbb{E}L(\boldsymbol{w}_{\text{avg,n}})$ respectively:

$$\mathbb{E}\Big(L(\boldsymbol{w}_n)\Big) \leq L(\boldsymbol{w}^*) + \frac{\alpha}{4}\text{Tr}(\boldsymbol{B}^{-1}\Sigma_g(\boldsymbol{w}^*))$$
$$\mathbb{E}\Big(L(\boldsymbol{w}_{\text{avg,n})}\Big) \leq L(\boldsymbol{w}^*) + \min\Big(\frac{1}{n+1}\text{Tr}(\boldsymbol{B}^{-1}\Sigma_g(\boldsymbol{w}^*)), \frac{\alpha}{2}\text{Tr}(\boldsymbol{B}^{-1}\Sigma_g(\boldsymbol{w}^*)\Big)$$

where $\boldsymbol{w}^*$ is the optimal point and $\Sigma_g(\boldsymbol{w}^*)$ is the covariance of the gradients at that point, and $\boldsymbol{H}$ is the (surrogate of) the Hessian matrix. This is to be contrasted to the pure SGD case, where the two losses are stated in Equation 7.26. Hence, in the limit $n \to \infty$, for both non-adaptive and adaptive methods $\mathbb{E}L(\boldsymbol{w}_{avg,n}))$ tends to the minimum whereas $\mathbb{E}(L(\boldsymbol{w}_n))$ does not[1]. Further, we note that the asymptotic limit is independent of $\boldsymbol{B}$ and α, so the result for adaptive optimisers and non-adaptive optimisers is identical for averaging in the asymptotic limit. Beyond the asymptotic equivalence, in fact we argue that with some mild assumptions, in some circumstances averaging iterates of adaptive methods could lead to even *better* expected loss bound during training. Formally, from Martens [2014], suppose we start averaging at a point $\boldsymbol{w}_0$ in the weight space that is different from minimum $\boldsymbol{w}^*$, the difference between the expected loss at the average of iterates $\mathbb{E}(\boldsymbol{w}_{avg,n})$ and the loss at minimum $L(\boldsymbol{w}^*)$ is less than:

$$\min\Big(\frac{1}{n+1}\text{Tr}(\boldsymbol{H}^{-1}\Sigma_g(\boldsymbol{w}^*)), \frac{\alpha}{2}\text{Tr}(\boldsymbol{B}^{-1}\Sigma_g(\boldsymbol{w}^*))\Big)$$
$$+ \min\Big(\frac{1}{(n+1)^2\alpha^2}||\boldsymbol{H}^{-1/2}\boldsymbol{B}(\boldsymbol{w}_0 - \boldsymbol{w}^*)||^2, \frac{1}{(n+1)\alpha}||\boldsymbol{B}^{1/2}(\boldsymbol{w}_0 - \boldsymbol{w}^*)||^2, 3L(\boldsymbol{w}_0)\Big) \tag{7.38}$$

[1] the discussion on pure SGD case is used for proof of Theorem 19.

For SGD, we have $\boldsymbol{B} = \boldsymbol{I}$, and thus Equation 7.38 reduces to

$$\mathbb{E}\Big(L(\boldsymbol{w}^{\text{SGD}}_{\text{avg,n}})\Big) - L(\boldsymbol{w}^*) \leq \frac{\text{Tr}(\boldsymbol{H}^{-1}\Sigma_g(\boldsymbol{w}^*))}{n+1} + \frac{||\boldsymbol{H}^{-1/2}(\boldsymbol{w}^{\text{SGD}}_0 - \boldsymbol{w}^*)||^2}{(n+1)^2\alpha^2} \tag{7.39}$$

On the other hand, adaptive methods has general update rule $\boldsymbol{w}_{n+1} \leftarrow \boldsymbol{w} - \alpha\boldsymbol{B}^{-1}\nabla L_{\boldsymbol{w}}$, where the inverse of the pre-conditioning matrix $\boldsymbol{B}^{-1}$ is generally intractable and is often approximated by $\text{diag}(\boldsymbol{B}^{-1})$ [Duchi et al., 2011]. Kingma and Ba [2014] argues that for Adam, the pre-conditioning matrix approximates the square-root of the diagonals of the Fisher Information Matrix, a positive-semidefinite surrogate of Hessian $\boldsymbol{H}$. Therefore, along these arguments, we assume that in Adam $\boldsymbol{B} \approx \boldsymbol{H}^{1/2}$. This yields,

$$\mathbb{E}\Big(L(\boldsymbol{w}^{\text{Adam}}_{\text{avg,n}})\Big) - L(\boldsymbol{w}^*) \leq \frac{\text{Tr}(\boldsymbol{H}^{-1}\Sigma_g(\boldsymbol{w}^*))}{n+1} + \frac{||\boldsymbol{w}^{\text{Adam}}_0 - \boldsymbol{w}^*||^2}{(n+1)^2\alpha^2} \tag{7.40}$$

and therefore, the difference compared to the SGD regret bound is on the *noise-independent term*. With same number of iterations n, we expect $||\boldsymbol{w}^{\text{Adam}}_0 - \boldsymbol{w}^*|| < ||\boldsymbol{w}^{\text{SGD}}_0 - \boldsymbol{w}^*||$ due to the faster convergence of Adam (In particular, as argued in the original Adam paper [Kingma and Ba, 2014], for sparse-bounded gradients, the regret and hence the convergence is reduced from $\mathcal{O}(\sqrt{Pn})$ to $\mathcal{O}(\log P\sqrt{n})$). Furthermore, as argued by Martens [2014], when $\boldsymbol{H}$ is ill-conditioned, $||\boldsymbol{w}^{\text{SGD}}_0 - \boldsymbol{w}^*||$ is likely to have large component in small eigenvalue directions, and the pre-conditioning by $\boldsymbol{H}^{-1/2}$ could lead to $||\boldsymbol{w}^{\text{Adam}}_0 - \boldsymbol{w}^*|| << ||\boldsymbol{H}^{-1/2}(\boldsymbol{w}^{\text{SGD}}_0 - \boldsymbol{w}^*)||$ and as a result, $\mathbb{E}\Big(L(\boldsymbol{w}^{\text{Adam}}_{\text{avg,n}})\Big) < \mathbb{E}\Big(L(\boldsymbol{w}^{\text{SGD}}_{\text{avg,n}})\Big)$.

7.7 Experimental Details

7.7.1 Image Classification Experiments

Hyperparameter Tuning In CIFAR experiments, we tune the base optimisers (i.e. SGD, Adam(W), Padam(W)) only, and assuming that the ideal hyperparameters in base optimisers apply to IA, and apply the same hyperparameter setting for the corresponding IA optimisers (i.e. SWA, Gadam, GadamX). For SGD, we use a base learning rate of 0.1 and use a grid searched initial learning rates in the range of $\{0.001, 0.01, 0.1\}$ and use the same learning rate for Padam, similar to the procedures suggested in Chen and Gu [2018]. For Adam(W), we simply use the default initial learning rate of 0.001 except in VGG-16, where we use initial learning rate of 0.0005. After the best learning rate has been identified, we conduct a further search on the weight decay, which we find often leads to a trade-off between the convergence speed and final performance; again we search on the base optimisers only and use the same value for the IA optimisers. For CIFAR experiments, we search in the range of $[10^{-4}, 10^{-3}]$, from the suggestions of Loshchilov and Hutter [2019]. For decoupled weight decay, we search the same range for the weight decay scaled by initial learning rate.

On ImageNet experiments, we conduct the following process. On the Wide Residual Network we use the settings recommended by Chrabaszcz et al. [2017], who conducted a thorough hyperparameter search: we set the learning rate at 0.03 and weight decay at 0.0001 for SGD/SWA and Padam, based on their searched optimal values. for AdamW/Gadam, we set decoupled weight decay at 0.01 and initial learning rate to be 0.001 (default Adam learning rate). For GadamX, we again use the same learning rate of 0.03, but since

the weight decay in GadamX is partially decoupled, we set the decoupled weight decay to 0.0003. On PRN-110, we follow the recommendations of the authors of He et al. [2016b] to set the initial learning rate for SGD, Padam and GadamX to be 0.1. For AdamW and Gadam, we again use the default learning rate of 0.001. Following the observation by Loshchilov and Hutter [2019] that smaller weight decay should be used for longer training (for the Pre-Residual-Network with 110 layers we train for 200 epochs), we set weight decay at 10^{-5} and decoupled weight decay at 0.0003 (GadamX)/0.001 (others) respectively, where applicable.

Overall, we do **not** tune adaptive methods (Adam and Gadam) as much (most noticeably, we usually fix their learning rate to 0.001), and therefore in particular the AdamW results we obtain may or may not be at their optimal performance. Nonetheless, the rationale is that by design, one of the key advantage claimed is that adaptive optimiser should be less sensitive to hyperparameter choice, and in this Chapter, the key message is that Gadam performs well, *despite AdamW, its base optimiser being rather crudely tuned.*

In all experiments, momentum parameters ($\beta = 0.9$) for SGD and $\{\beta_1, \beta_2\} = \{0.9, 0.999\}$ and $\epsilon = 10^{-8}$ for Adam and its variants are left at their respective default values. For all experiments unless otherwise stated, we average once per epoch. We also apply standard data augmentation (e.g. flip, random crops) and use a batch size of 128 for all experiments conducted.

7.7.2 Language Modelling Experiments

In the language modelling experiments, we use the codebase provided by `https://github.com/salesforce/awd-lstm-lm`. For ASGD, we use the hyperparameters recommended by Merity et al. [2017] and set the initial

learning rate to be 30. Note that in language experiments, consistent with other findings decoupled weight decay seems to be not as effective L_2, possibly due to LSTM could be more well-regularised already, and that BN, which we argue to be central to the efficacy of decoupled weight decay, is not used in LSTM. Thus, for this set of experiments we simply use Adam and Padam as the iterates for Gadam and GadamX. For Adam/Gadam, we tune the learning rate by searching initial learning rate in the range of $\{0.0003, 0.001, 0.003, 0.01\}$ and for Padam and GadamX, we set the initial learning rate to be 1 and partially adaptive parameter $p = 0.2$, as recommended by the authors Chen and Gu [2018]. We further set the weight decay to be their recommended value of $1.2E{-}6$. For the learning rate schedule, we again follow Merity et al. [2017] for a piecewise constant schedule, and decay the learning rate by a factor of 10 at the $\{100, 150\}$-th epochs for all experiments without using iterate averaging. For experiments with iterate averaging, instead of decaying the learning rate by half before averaging starts, we keep the learning rate constant thoughout to make our experiment comparable with the ASGD schedule. We run all experiments for 200 (instead of 500 in Merity et al. [2017]) epochs.

Learning Rate Schedule As discussed in the main text, the experiments shown in Table 7.2 and Fig. 7.10 are run with constant schedules (except for Padam). Padam runs with a step decay of factor of 10 at $\{100, 150\}$-th epochs. However, often even the adaptive methods such as Adam are scheduled with learning rate decay for enhanced performance. Therefore, we also conduct additional scheduled experiments with Adam, where we follow the same schedule of Padam.

7.8 Experiments

In this section, we test our proposed algorithms in multiple vision and language processing tasks. For robustness we run each experiment 3 times and we report the mean and standard deviation (in brackets in all of the tables). We open-source our code[2] to the community.

7.8.1 Image Classification on CIFAR Data-sets

Here apart from SGD, Adam(W), Padam, we sometimes also include Padam with decoupled weight decay, which we term *PadamW* on various architectures (VGG-16, Preactivated ResNet (PRN) and ResNeXt [Simonyan and Zisserman, 2015, He et al., 2016b, Xie et al., 2017]) on CIFAR data-sets [Krizhevsky et al., 2009], and show the evolution of test accuracy against number of epochs on CIFAR-100 in Figure 7.8 and the final value at the end of training in Table 7.2. The corresponding results on CIFAR-10 are in Table 7.3. As AdamW always outperform Adam in our experiments, the curves for the latter are omitted. Similarly, we also only show either Padam or PadamW in our figures, whichever performs better in terms of final accuracy. To ensure our baselines are fairly tuned, we also include the results reported in the previous works in Table 7.1. The results show that optimisers with IA (SWA, Gadam and GadamX) invariably improve over their counterparts without IA, and in all cases GadamX delivers the strongest test performance by a considerable margin over baselines (except for PRN-110, where any difference between SWA and GadamX is not immediately significant, although GadamX does have a smaller standard deviation across different seeds and converge faster). Even without any modification to the adaptivity, it is re-

[2] https://github.com/diegogranziol/Gadam

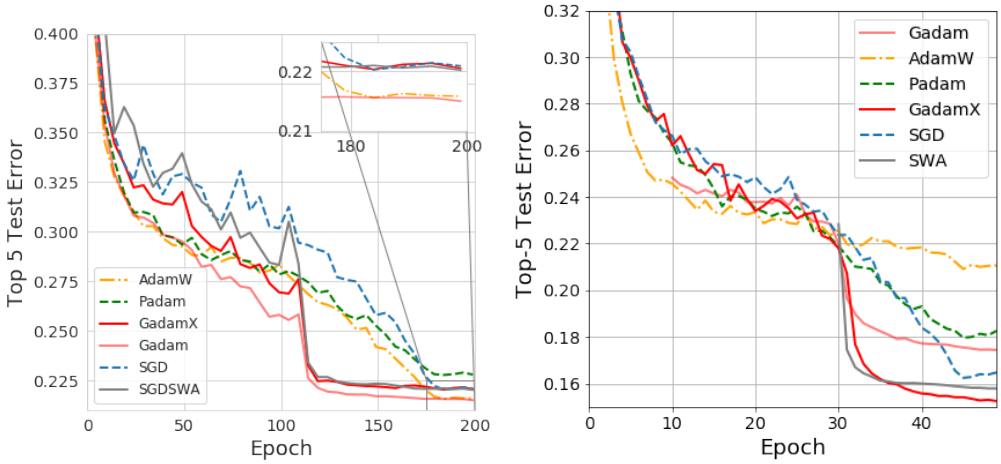
Table 7.1: Baseline Results from Previous Works

Network	Optimiser	Accuracy/Perplexity	Reference
CIFAR-100			
VGG-16	SGD	73.80	[Huang and Wang, 2018]
VGG-16	FGE	74.26	[Izmailov et al., 2018]
PRN-164	SGD	75.67	[He et al., 2016b]
PRN-110	SGD	76.35	online repository**
ResNet-164	FGE	79.84	[Izmailov et al., 2018]
ResNeXt-29	SGD	82.20	[Xie et al., 2017]
ResNeXt-29	SGD	81.47	[Bansal et al., 2018]
CIFAR-10			
VGG-19	SGD	93.34	online repository**
VGG-16	SGD	93.90	[Huang and Wang, 2018]
PRN-110	SGD	93.63	[He et al., 2016b]
PRN-110	SGD	95.06	online repository**
ImageNet 32×32			
WRN-28-10	SGD	59.04/81.13*	[Chrabaszcz et al., 2017]
Modified WRN	SGD	60.04/82.11*	[McDonnell, 2018]
PTB			
LSTM 3-layer	NT-ASGD	61.2/58.8***	[Merity et al., 2017]

Notes:
* Top-1/Top-5 Accuracy
** Link: https://github.com/bearpaw/pytorch-classification
*** Validation/Test Perplexity

markable that in all cases Gadam outperforms fine-tuned SGD and Padam - this seems to suggest that solutions found by adaptive optimisers are not necessarily inferior in their generalisation capability, in contrast to some common beliefs. Indeed, any generalisation gap seems to be closed by the simply using iterate averaging and an appropriately implemented weight decay.

We report the testing performance of VGG-16 and PRN-110 on CIFAR-10 in Fig. 7.9 and Table 7.3. Perhaps due to the fact that CIFAR-10 poses a simpler problem compared to CIFAR-100 and ImageNet in the main text,



(a) PRN-110 (b) WRN-28-10

Figure 7.7: Top-5 Test Error on ImageNet 32×32.

Table 7.2: Top-1 Test Accuracy on CIFAR-100 Data-set.

Architecture	optimiser	Test Accuracy
VGG-16	SGD	74.15±0.06
	SWA	74.57±0.27
	Adam(W)	73.26±0.30
	Padam(W)	74.56±0.19
	Gadam	75.73±0.29
	GadamX	**76.85**±0.08
PRN-110	SGD	77.22±0.05
	SWA	**77.92**±0.36
	Adam(W)	75.47±0.21
	Padam(W)	77.30±0.11
	Gadam	77.37±0.09
	GadamX	**77.90**±0.21
ResNeXt-29	SGD	81.47±0.17
	SWA	82.95±0.28
	Adam(W)	80.16±0.16
	Padam(W)	82.37±0.35
	Gadam	82.13±0.20
	GadamX	**83.27**±0.11

the convergence speeds of the optimisers differ rather minimally. Nonetheless, we find that GadamX still outperforms all other optimisers by a non-trivial

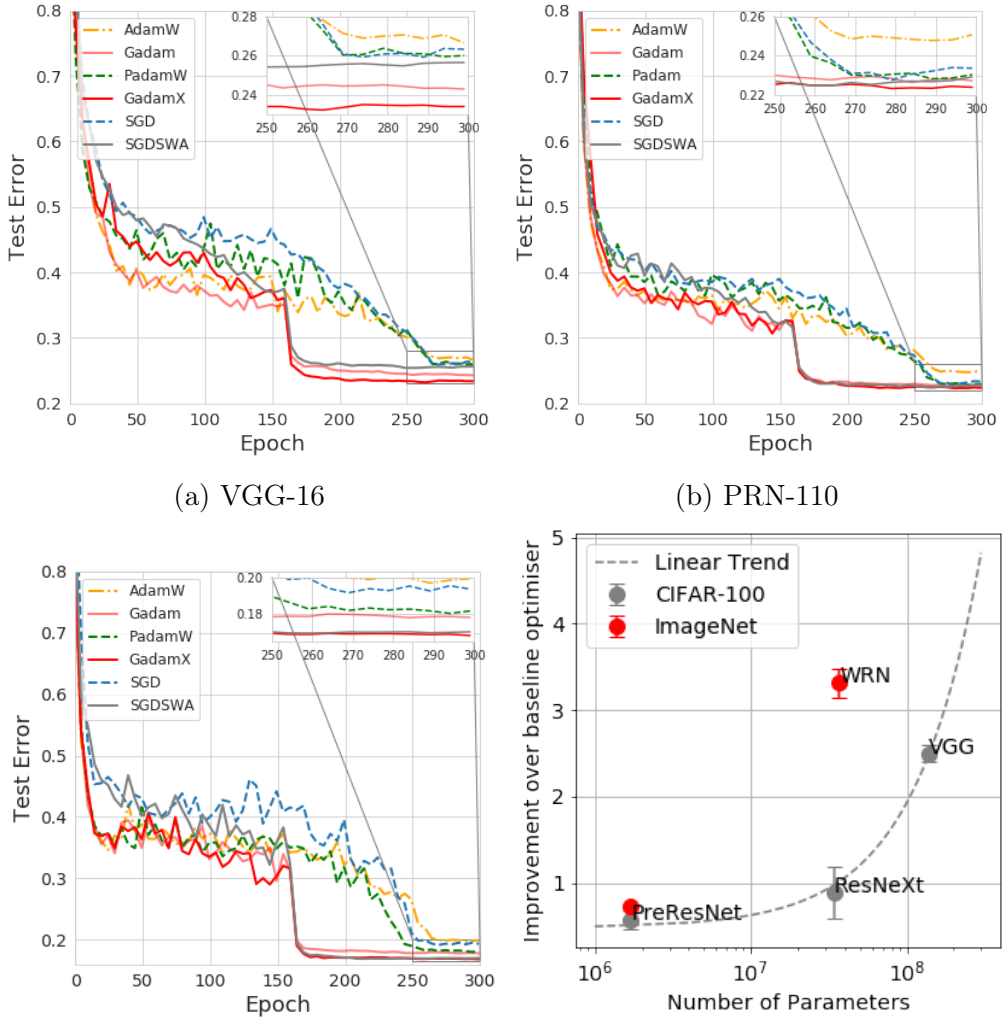
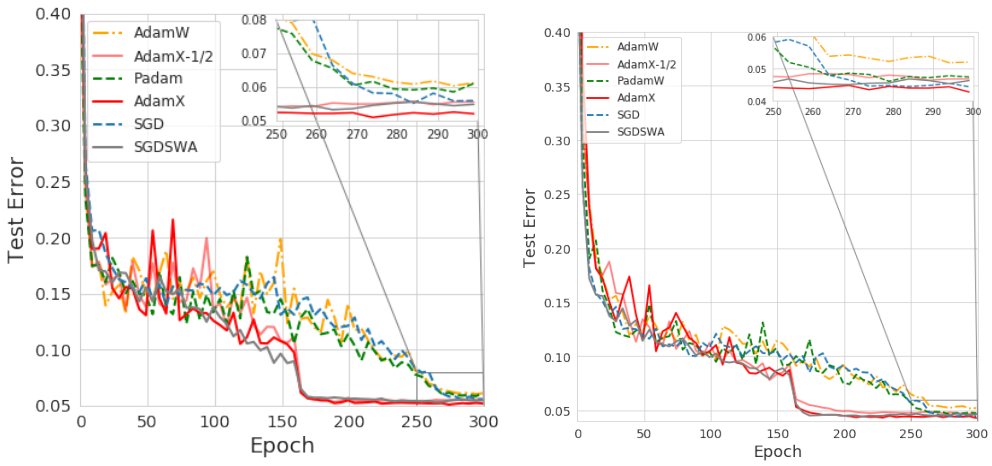


Figure 7.8: Test error on CIFAR-100 (a-c) and test improvement of best IA over its base optimiser against number of parameters (d).

margin in terms of final test accuracy.

7.8.2 Image Classification on ImageNet 32×32

We conduct experiments on the ImageNet dataset with the resolution of each picture down-sampled to 32×32 [Chrabaszcz et al., 2017] and results



(a) VGG-16 (b) PRN-110

Figure 7.9: Test Error on CIFAR-10.

Table 7.3: Top-1 Test Accuracy on CIFAR-10 Data-set

Architecture	optimiser	Test Accuracy
VGG-16	SGD	94.14±0.37
	SWA	94.69±0.36
	Adam(W)	93.90±0.11
	Padam(W)	94.13±0.06
	Gadam	94.62±0.15
	GadamX	**94.88**±0.03
PRN-110	SGD	95.40±0.25
	SWA	95.55±0.12
	Adam(W)	94.69±0.14
	Padam(W)	95.28±0.13
	Gadam	95.27±0.02
	GadamX	**95.95**±0.06

are shown in Figure 7.7. Similar to the CIFAR results, our proposed algorithms perform competitively. In PRN-110, again we see similar behaviour compared to the CIFAR experiments that IA only leads to modest improvements in test performance. This is unsurprising, as this architecture has the smallest extent of over-parameterisation if we use number of parameters P

as a crude estimate of the model complexity (actually in ImageNet experiments, we almost have $P = N$, which might also explain the out-performance of Adam-based over SGD-based optimisers, as overfitting and the resultant need for regularisation might be less important in this context). On the other hand, in WideResNet (WRN) [Zagoruyko and Komodakis, 2016] experiments, Gadam does not outperform our SGD implementation [3] perhaps due the default learning rate in AdamW/Gadam we use. Despite that, Gadam greatly improves upon AdamW, and already posts a performance stronger than baseline in literature with identical [Chrabaszcz et al., 2017] and improved [McDonnell, 2018] setups. On the other hand, GadamX performs very strongly, posting an out-performance of more than 3% compared to the baseline in Chrabaszcz et al. [2017] in Top-5 accuracy. In line with our noisy quadratic model and the results from Theorem 18, we observe a near linear relation between P and benefit of IA measured in terms of test improvement compared to the SGD baseline, shown in Figure 7.8d. Whilst we acknowledge that the variety in optimal hyper-parameter setup between the best runs may be significant and probably more data points are needed to conclusively and rigorously establish the linear relation, it is at least encouraging to see some agreement with our simplistic model in real networks. We also find that a similar pattern also holds in previous work [Izmailov et al., 2018] despite of using different architectures compared to ours.

7.8.3 Word-level Language Modelling on PTB

Lastly, we also run word-level language modelling using a 3-layer Long-short Term Memory (LSTM) model [Gers et al., 1999] on the Penn Treebank (PTB) data-set [Marcus et al., 1993] and we present the results in Table 7.5 and

[3] The SGD baseline we report is much stronger (2% higher) than the literature.

Table 7.4: Test Accuracy on ImageNet 32×32 Data-set.

Architecture	optimiser	Test Accuracy	
		Top-1	Top-5
PRN-110	SGD	54.27±0.36	77.96±0.08
	SWA	54.37±0.18	78.04±0.14
	AdamW	54.84±0.20	**78.43**±0.11
	Padam	53.71±0.15	77.31±0.09
	Gadam	**54.97**±0.12	**78.48**±0.02
	GadamX	54.45±0.49	77.91±0.27
WRN-28-10	SGD	61.33±0.11	83.52±0.14
	SWA	62.32±0.13	84.23±0.05
	AdamW	55.51±0.19	79.09±0.33
	Padam	59.65±0.17	81.74±0.16
	Gadam	60.50±0.19	82.56±0.13
	GadamX	**63.04**±0.06	**84.75**±0.03

Figure 7.10. Remarkably, Gadam even achieves an better result than the baseline NT-ASGD in Merity et al. [2017], although the latter runs an additional 300 epochs on an identical network. Note that since, by default, the ASGD uses a constant learning rate, we do *not* schedule the learning rate except Padam which requires scheduling to converge. Also, for consistency, we use a manual trigger to start averaging at the 100th epoch for ASGD (which actually outperforms the NT-ASGD variant).

Table 7.5: Validation and Test Perplexity on Word-level Language Modelling.

Data-set	optimiser	Perplexity	
		Validation	Test
PTB	ASGD	64.88±0.07	61.98±0.19
	Adam	65.96±0.08	63.16±0.24
	Padam	65.69±0.07	62.15±0.12
	Gadam	**61.35**±0.05	**58.77**±0.08
	GadamX	63.49±0.19	60.45±0.04

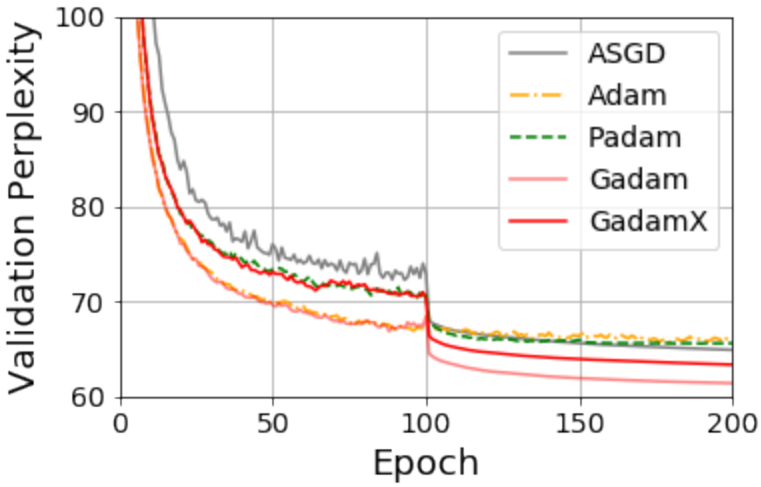


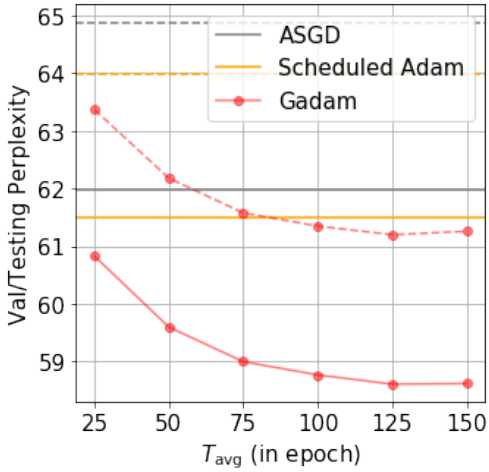
Figure 7.10: Validation perplexity of 3-layer LSTM on PTB word-level modelling against the number of epochs.

7.8.4 Effect of T_{avg}

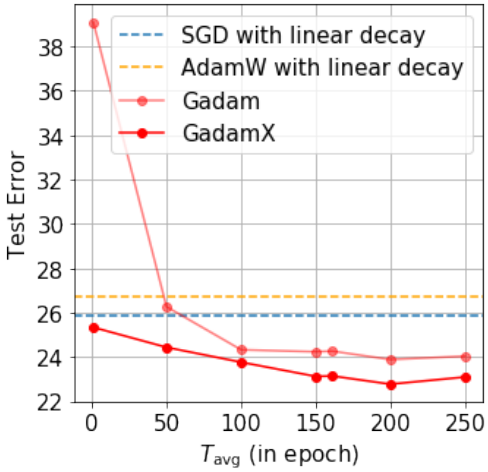
In Gadam(X), we need to determine when to start averaging (T_{avg} in Algorithm 9), and here we investigate the sensitivity of Gadam(X) to this hyperparameter. We use a range of T_{avg} for a number of different tasks and architectures (Fig. 7.11 and Table 7.6), including extreme choices such as $T_{\text{avg}} = 0$ (start averaging at the beginning). We observe that for any reasonable T_{avg}, Gadam(X) always outperform their base optimisers with standard learning rate decay, and tuning T_{avg} yields even more improvements over the heuristics employed in the main text, even if selecting any sensible T_{avg} already can lead to a promising performance over standard learning rate decay.

7.8.5 Gadam and Lookahead

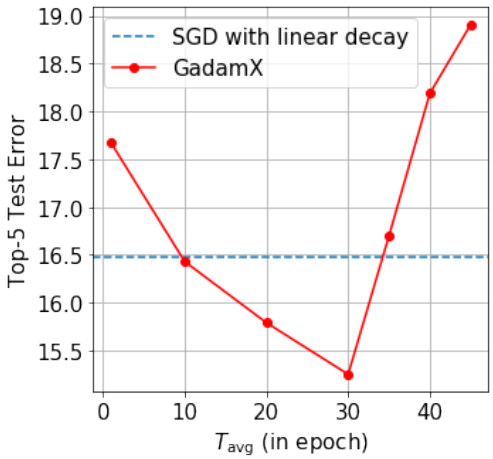
Lookahead [Zhang et al., 2019b] is a very recent attempt that also features weight space averaging in order to achieve optimisation and generalisation benefits. However, instead of using simple averaging in our proposed al-



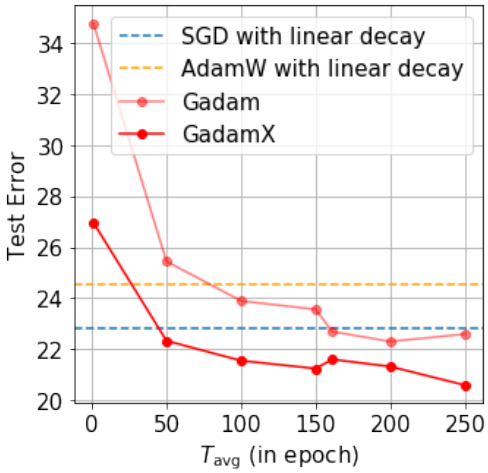
(a) 3-Layer LSTM on PTB†.



(b) VGG-16 on CIFAR-100



(c) WRN-28-10 on ImageNet 32×32



(d) PRN-110 on CIFAR-100

Figure 7.11: Effect of different T_{avg} on the performance of various tasks and architectures. †: In (a), *dashed lines* denote validation and *solid lines* denote test perplexities.

gorithms, Lookahead maintains different update rules for the *fast* and *slow* weights, and uses exponentially moving average to update the parameters. In this section, we both comment on the key theoretical differences between Gadam and Lookahead and make some preliminary practical comparisons. We also offer an attempt to bring together the *optimisation* benefit of Looka-

Table 7.6: Best results obtained from tuning T_{avg}

Architecture	Optimiser	Test Acc./Perp.
CIFAR-100		
VGG-16	Gadam	76.11
	GadamX	77.22
PRN-110	Gadam	77.41
	GadamX	79.41
ImageNet 32×32		
WRN-28-10	GadamX	84.75
PTB		
LSTM	Gadam	58.61

head and the *generalisation* benefit of Gadam, with promising preliminary results.

7.8.5.1 Major Differences between Gadam and Lookahead

Averaging Method: Lookahead opts for a more complicated averaging scheme: they determine the *fast-* and *slow* -varying weights during optimisation, and maintains an EMA to average the weight. On the other hand, Gadam uses a more straightforward simple average. As we discussed in the main text, EMA is more theoretically justified during the initial rather than later stage of training. This can also be argued from a Bayesian viewpoint following Maddox et al. [2019], who argued that iterates are simply the draws from the posterior predictive distribution of the neural network, where as averaging leads to a rough estimation of its posterior mean. It is apparent that if the draws from this distribution are *equally* good (which is likely to be the case if we start averaging only if validation metrics stop improving), assigning the iterates with an exponential weight just based on when they are drawn constitutes a rather arbitrary prior in Bayesian sense.

Averaging Frequency: Lookahead averages every iteration whereas in Gadam, while possible to do so as well, by default averages much less frequently. We detail our rationale for this in Section 7.8.6.

Starting Point of Averaging: While Lookahead starts averaging at the beginning of the training, Gadam starts averaging either from a pre-set starting point or an automatic trigger (for GadamAuto). While authors of Lookahead [Zhang et al., 2019b] argue that starting averaging eliminates the hyperparameter on when to start averaging, it is worth noting that Lookahead also introduces two additional hyperparameters α and k, which are non-trivially determined from grid search (although the authors argue that the final result is not very sensitive to them).

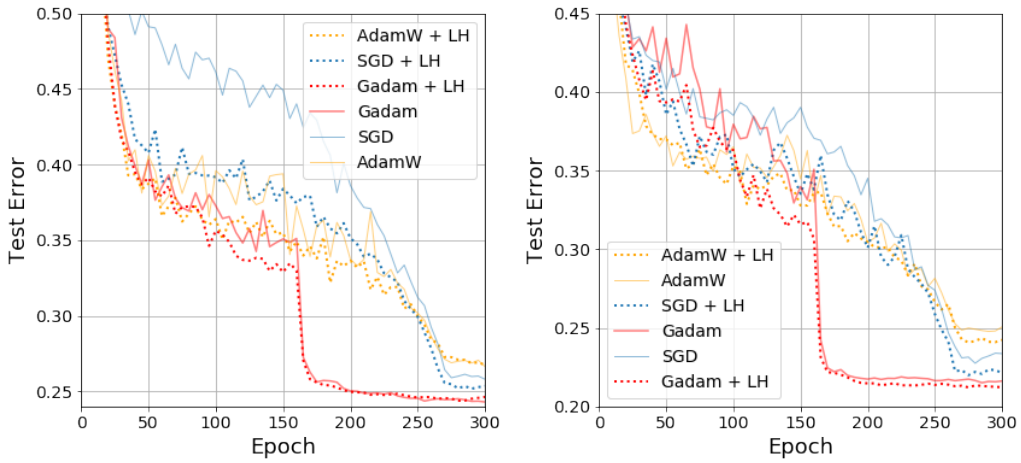
We believe the difference here is caused by the different design philosophies of Gadam and Lookahead: by using EMA and starting averaging from the beginning, Lookahead benefits from faster convergence and some generalisation improvement whereas in Gadam, since the averages of iterates are not used during training to promote independece between iterates, Gadam does not additionally accelerate optimisation but, by our theory, should generalise better. As we will see in the next Section, this theoretical insight is validated by the experiments and leads to combinable benefits.

Empirical Comparison

We make some empirical evaluations on CIFAR-100 data-set with different network architectures, and we use different base optimiser for Lookahead. For all experiments, we use the author-recommended default values of $k = 5$ (number of lookahead steps) and $\alpha = 0.5$. We focus on the combination of Lookahead and adaptive optimisers, as this is the key focus of this Chap-

ter, although we do include results with Lookahead with SGD as the base optimiser.

We first test AdamW and SGD with and without Lookahead and the results are in Figure 7.12. Whilst SGD + LH outperforms SGD in final test accuracy by a rather significant margin in both architectures, Lookahead does not always lead to better final test accuracy in AdamW (although it does improve the convergence speed and reduce fluctuations in test error during training, which is unsurprising as EMA shares similar characteristics with IA in reducing sensitivity to gradient noise). On the other hand, it is clear that Gadam delivers both more significant and more consistent improvements over AdamW, both here and in the rest of the chapter.



(a) VGG-16 (b) PRN-110

Figure 7.12: Test accuracy of Lookahead in CIFAR-100 against number of epochs.

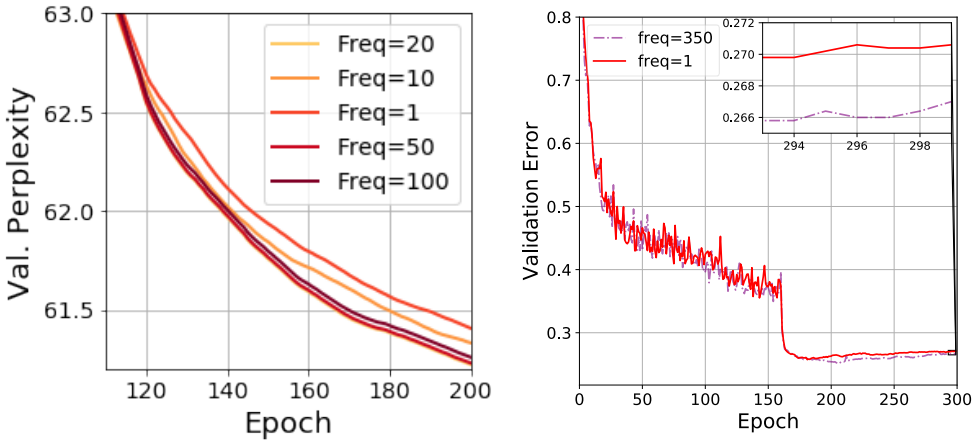
Nonetheless, we believe that Lookahead, being an easy-to-use plug-in optimiser that clearly improves convergence speed, offers significant combinable potential with Gadam, which focuses on generalisation. Indeed, by using Lookahead *before* the 161st epoch where we start IA, and switching to IA

after the starting point, we successfully combine Gadam and LH into a new optimiser which we term Gadam + LH. With reference to Figure 7.12, in VGG-16, Gadam + LH both converges at the fastest speed in all the optimisers tested and achieves a final test accuracy only marginally worse than Gadam (but still stronger than all others). On the other hand, in PRN-110, perhaps due to the specific architecture choice, the initial difference in convergence speed of all optimisers is minimal, but Gadam + LH clearly performs very promisingly in the end: it is not only stronger than our result without Lookahead in Figure 7.12(b), but also, by visual inspection, significantly stronger than the SGD + LH results on the same data-set and using the same architecture reported in the original Lookahead paper [Zhang et al., 2019b].

7.8.6 Effect of Frequency of Averaging

While we derive the theoretical bounds using Polyak-style averaging on every *iteration*, practically we average *much* less: we either average once per *epoch* similar to Izmailov et al. [2018], or select a rather arbitrary value such as averaging once per 100 iterations. The reason is both practical and theoretical: averaging much less leads to significant computational savings, and at the same time as we argued in Sections 7.5 and 7.5.6, on more independent iterates the benefit from averaging is better, because our theoretical assumptions on independence are more likely met in these situations. In this case, averaging less causes the iterates to be further apart and more independent, and thus fewer number of iterates is required to achieve the similar level of performance if less independent iterates are used. We verify this both on the language and the vision experiments using the identical setup as the main text. With reference to Figure 7.13(a), not only is the final perplexity very

insensitive to averaging frequency (note that the y-axis scale is very small), it is also interesting that averaging *less* actually leads to a slightly better validation perplexity compared to schemes that, say, average every iteration. We see a similar picture emerges in Figure 7.13(b), where the despite of following very close trajectories, averaging every iteration gives a slightly worse testing performance compared to once an epoch and is also significantly more expensive (with a NVIDIA GeForce RTX 2080 Ti GPU, each epoch of training takes around 10s if we average once per epoch but averaging every iteration takes around 20s).



(a) LSTM on PTB

(b) VGG-16 on CIFAR-100

Figure 7.13: Effect of different averaging frequencies on validation perplexity of Gadam on representative (a) Language and (b) Image classification tasks. `Freq=`n suggests averaging once per n iterations. `freq=350` in (b) is equivalently averaging once per *epoch*.

7.8.7 GadamAuto Experiments

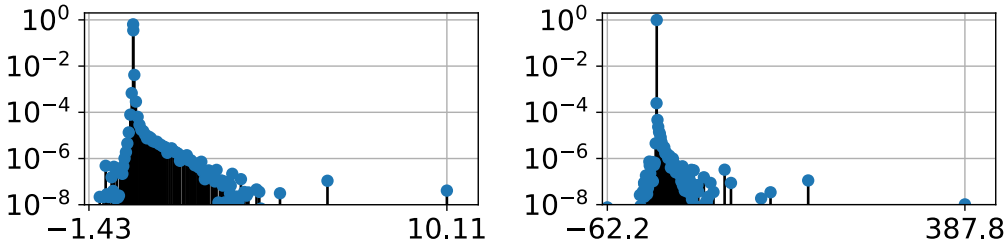
Here we conduct preliminary experiments on GadamAuto, a variant of Gadam that uses a constant learning rate schedule and automatically determines the starting point of averaging and training termination - this is desirable as the

optimiser both has fewer hyperparameters to tune and trains faster. We use VGG-16 network on CIFAR-100. For all experiments, we simply use a flat learning rate schedule. The results are shown in Table 7.7. We use a patience of 10 for both the determination of the averaging activation and early termination. We also include SWA experiments with SGD iterates.

Table 7.7: GadamAuto Test Performance at Termination.

Optimiser	Data-set	Test Accuracy
Gadam-Auto	CIFAR-100	75.39
SWA-Auto	CIFAR-100	73.93

It can be seen that while automatic determination for averaging trigger and early termination work well for Gadam (GadamAuto posts a performance only marginally worse than the manually tuned Gadam), they lead to a rather significant deterioration in test in SWA (SWA-Auto performs worse than tuned SWA, and even worse than tuned SGD. See Table 7.2). This highlights the benefit of using adaptive optimiser as the base optimiser in IA, as the poor performance in SWA-Auto is likely attributed to the fact that SGD is much more hyperparameter-sensitive (to initial learning rate and learning rate schedule, for example. SWA-Auto uses a constant schedule, which is sub-optimal for SGD), and that validation performance often fluctuates more during training for SGD: SWA-Auto determines averaging point based on the number of epochs of validation accuracy stagnation. For a noisy training curve, averaging might be triggered too early; while this can be ameliorated by setting a higher patience, doing so will eventually defeat the purpose of using an automatic trigger. Both issues highlighted here are less serious in adaptive optimisation, which likely leads to the better performance of GadamAuto. Nonetheless, the fact that scheduled Gadam still outperforms GadamAuto suggests that there is still ample room for improve-



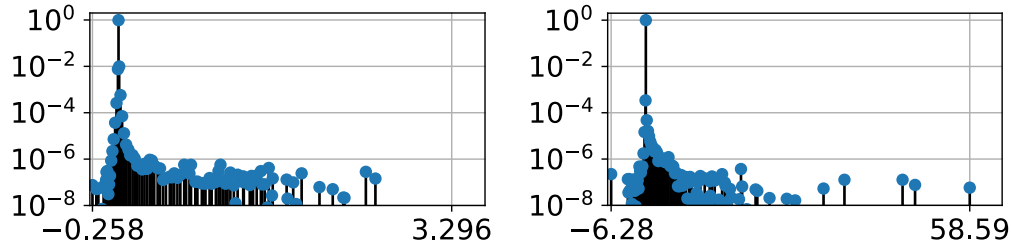
(a) $Val = 73.5$, SGD (b) $Val = 75.8$, Gadam

Figure 7.14: Hessian spectrum for VGG-16BN after 300 epochs of SGD on the CIFAR-100 dataset, for various optimisation algorithms [SGD, Gadam], batch norm train mode.

ment to develop a truly automatic optimiser that performs as strong as or even stronger than tuned ones. One desirable alternative we propose for the future work is the integration of *Rectified Adam* [Liu et al., 2019], which is shown to be much more insensitive to choice of hyperparameter even compared to Adam.

7.9 Sharpness and Adaptive optimisation

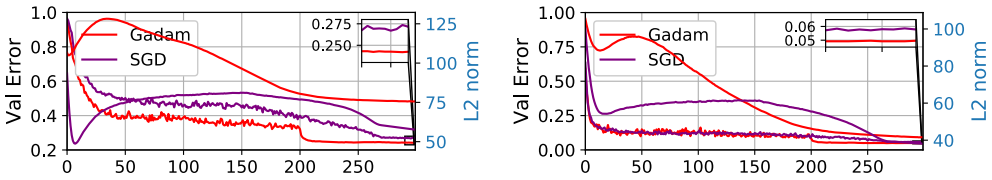
Given the extensive discussions on the generalisability of "flat minima" Hochreiter and Schmidhuber [1997], Jastrzkbski et al. [2017], Wu et al. [2018] and used as a specific motivator for the use of IA in conjunction with SGD Izmailov et al. [2018] we consider whether higher performing solutions found by Gadam are flatter than those of SGD. We use a decoupled weight decay [Loshchilov and Hutter, 2019] of 0.35/0.25 and a learning rate of 0.0005 For SGD we use a weight decay of $3/5 \times 10^{-4}$ and a learning rate of 0.1. We plot the validation accuracy curve for CIFAR-100 in Figure 7.16a, Gadam clearly generalises better than SGD. As shown in Figures 7.14a and 7.14b, the spectral norm of the better performing Gadam solutions is almost 40× larger than the SGD solution, the Frobenius norm of Gadam is 0.02 as opposed to 0.0001



(a) $Val = 94.3$, SGD (b) $Val = 95.1$, Gadam

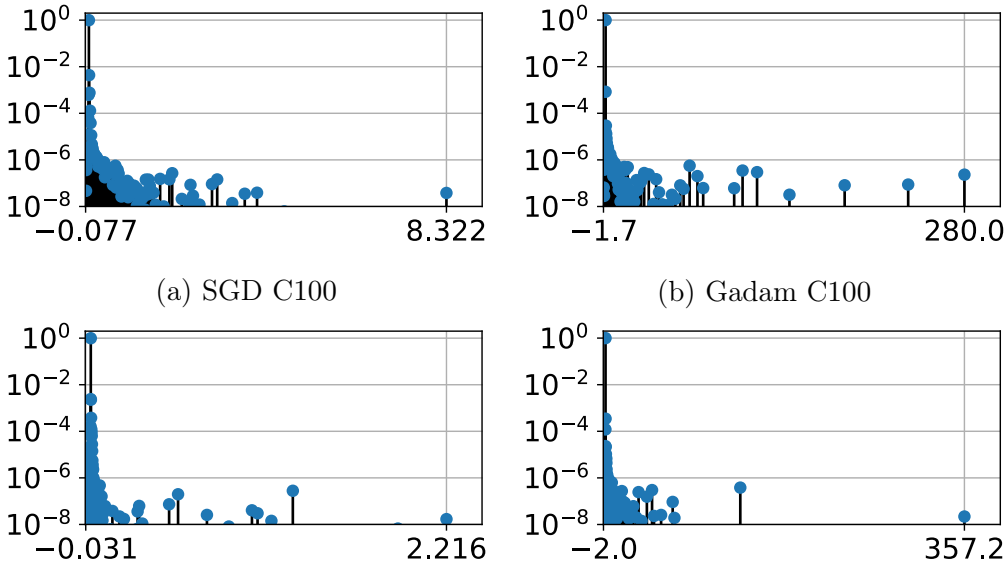
Figure 7.15: Hessian spectrum for VGG-16BN after 300 epochs of SGD on the CIFAR-10 dataset, for various optimisation algorithms [SGD, Gadam], batch norm train mode.

for SGD. Both solutions give similar training performance, with Gadam 99.81 and SGD 99.64. For CIFAR-10 although the generalisation gap is smaller, we see a similar picture, as shown in Figure 7.15. The Frobenius norm of the Gadam solution is 1.55×10^{-3} as opposed to 1.43×10^{-5}. When batch normalisation is set to evaluation mode, we find a very similar picture. With the spectral norm still 35× larger for the better performing solution. The Frobenius norm is also 0.04 instead of 1.3×10^{-5} for CIFAR-100 and 0.012 instead of 7.7×10^{-7} for CIFAR-10. As shown in Figure 7.16 the Gadam solution always finds a higher weight norm. Many works consider Neyshabur et al. [2017] consider that both sharpness and norm must be considered to explain generalisation. Here we note that the better generalising solutions have both higher norm and sharper spectrum.



(a) CIFAR-100 (b) CIFAR-10

Figure 7.16: Test Error & $L2$ norm of the VGG-16 of SGD/Gadam on the CIFAR-10/100 datasets.



(a) SGD C100 (b) Gadam C100

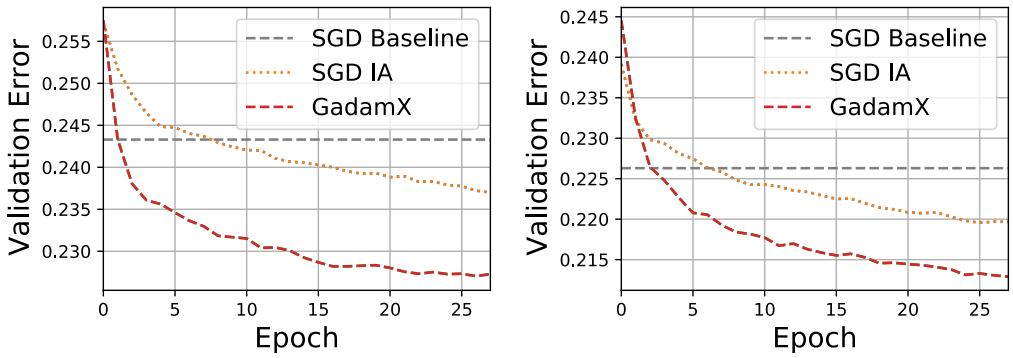
(c) SGD C10 (d) Gadam C10

Figure 7.17: Hessian spectrum for VGG-16BN after 300 epochs of SGD on the CIFAR-100 dataset, for various optimisation algorithms [SGD, Gadam], batch norm evaluation mode.

7.10 ImageNet

In the previous set of experiments, we specifically kept the learning rate schedule of Gadam/GadamX at the default values of the base optimiser to show the performance even with crude tuning. We also kept the linear SGD learning rate schedule as a baseline from Izmailov et al. [2018] in order to allow for strong experimental comparison to previous works. However for large scale experiments, such as the full ImageNet dataset, where typically practitioners use step learning rate decay rates, it is pertinent to consider whether IA and specifically adaptive optimisation with IA can bring generalisation benefits. In order to investigate this, we run the well known ResNet-50 and ResNet-101 architectures [He et al., 2016a] on the ImageNet dataset, with the typical learning rate of 0.1 (decayed every 30 epochs by a factor of 10) and

weight decay of 0.0001 for 90 epochs. We grid search both stochastic gradient descent with IA (SGD IA/SWA both terms are equivalent) and GadamX which is our strongest performing novel optimiser in terms of test performance, on both the initial and final learning rate, increasing and decreasing by factors of 3 until the best performance is reached. We also experiment with tuning the epsilon coefficient in Adam from its default value of 10^{-8} to 10^{-4} as argued in Chapter 6, which we find boosts the top-1 error by around 0.1%. We show the best run of both SGD IA and GadamX for both networks in Figure 7.18 against the final iterate validation error of the SGD baseline. We note that both IA networks improve upon the baseline, that the improvement is monotonic with increasing number of IA epochs and crucially that *GadamX delivers significantly improved performance on SGD with IA*.

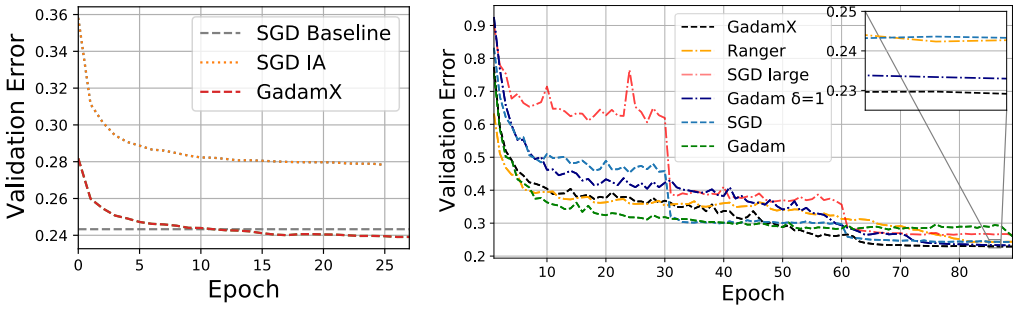


(a) ResNet-50 (b) ResNet-101

Figure 7.18: **Don't use SGD and step scheduling, use Adaptive optimisers with Iterate Averaging!** Final 30 epochs top-1 validation error on ImageNet. showing the improvement of both SGD with Iterate Averaging (SGD IA) and our proposed GadamX optimiser an SGD step-schedule baseline

Of particular interest are the hyper-parameters required to get these improved results. For the best performing GadamX results we increase the initial learning rate to 0.3, whereas for SGD IA we reduce the terminal learning

rate to 0.003. We show the results from our first experiments where we keep the same learning rate as was found to be optimal for ImageNet-32 in Figure 7.19a. We note that for SGD IA, we significantly underperform the baseline, whereas GadamX marginally beats the baseline. This indicates the robustness to tuning which was a main motivator for using (partially adaptive) optimisers in the first place.



(a) Single Shot (b) Convergence at a Cost

Figure 7.19: ResNet-50 ImageNet Validation Errors. (a) initial run using previous optimal hyper-parameters for ImageNet-32 for both SGD IA and GadamX against the SGD baseline, result demonstrates the lack of tuning required for strong GadamX results. (b) Various optimisers validation error, showing that tuned GadamX does the best, increasing the SGD learning rate only decreases the performance and that Gadam never quite performs as well as GadamX

7.10.1 Need for Speed: Convergence Comes at a Cost

We show a set of runs and their corresponding validation error for the ResNet-50 in Figure 7.19b. We also run an improved variant of the Lookahead optimiser, which uses Gradient centralisation [Yong et al., 2020], which we find despite converging faster, despite extensive tuning does not beat the SGD validation performance. We also run Gadam with a large initial learning rate $\alpha = 0.5$ and a damping value of 1 (shown as Gadam $\delta = 1$), which despite

converging slower than GadamX, does not quite catch up to its generalisation performance (but beats all other optimisers). Here we note that running Gadam with smaller learning rates and smaller damping, leads to faster optimisation, but smaller generalisation. Hence it seems like the price paid for fast convergence is poorer final test performance, even when using decoupled weight decay and IA. We also note that increasing the initial learning rate for SGD (to a learning rate of 0.3 labelled SGD large in the Figure), decreases both the convergence speed and the test performance. This is a somewhat unexpected effect, not explored in the previous set of experiments, which seems to be unique to adaptive optimisers with IA. *By managing to stably train at large initial learning rates (albeit slowly), it seems like adaptive optimisers in conjunction with IA are able to reach solutions with greater generalisation.*

8 | Concluding Remarks

In this thesis we develop a spectral approach to many areas of machine learning. We initially focus on speed ups in Bayesian algorithms by combining stochastic trace estimation and the method of Maximum entropy. We later extend this technique to the open problems of cluster counting and graph similarity. We then take the ideas of moment-matched approximations to learn the spectra of Deep Neural Network Hessians and positive-definite approximations thereof. We develop an open-source software package with simple loss visualisation and second-order optimisation functionality. We then develop a theoretical model using a spiked field dependent random matrix model of the fluctuations of the Hessian spectrum due to mini-batching. We use our software package to validate our theoretical predictions and show that as simple consequences of our theory, that we derive linear scaling and square-root scaling rules for the learning rate of Stochastic Gradient Descent and adaptive-gradient optimisers as we increase the batch size. This holds up to a threshold is reached, beyond which no appreciable learning rate increase is possible. Further consequences of our framework involve understanding some of the generalisation gap between adaptive and non-adaptive optimisers. We use this framework to justify certain technqiues already known in the literature, which have been considered heuristic until this work. As a final usage of our software package we investigate the relationships between spectral sharpness and generalisation and find that removing weight decay, which increases model complexity and overfits to the training data, decreases spectral sharpness. We prove this result in the limit and inspired by this idea develop an adaptive algorithm that finds sharp minima that generalise even better than those found by Stochastic Gradient Descent.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

Gernot Akemann, Jinho Baik, and Philippe Di Francesco. *The Oxford handbook of random matrix yheory*. Oxford University Press, 2011.

Anonymous. Towards understanding the true loss surface of deep neural networks using random matrix theory and iterative spectral methods. In *Submitted to International Conference on Learning Representations*, 2020. under review.

Zhaojun Bai, Gark Fahey, and Gene Golub. Some large-scale matrix computation problems. *Journal of Computational and Applied Mathematics*, 74 (1-2):71–89, 1996.

Zhi Dong Bai. Convergence rate of expected spectral distributions of large random matrices part i: Wigner matrices. In *Advances In Statistics*, pages 60–83. World Scientific, 2008.

Jinho Baik and Jack W Silverstein. Eigenvalues of large sample covariance matrices of spiked population models. *arXiv preprint math/0408165*, 2004.

Jinho Baik, Gérard Ben Arous, Sandrine Péché, et al. Phase transition of the largest eigenvalue for nonnull complex sample covariance matrices. *The Annals of Probability*, 33(5):1643–1697, 2005.

K Bandyopadhyay, Arun K Bhattacharya, Parthapratim Biswas, and DA Drabold. Maximum entropy and the problem of moments: A stable algorithm. *Physical Review E*, 71(5):057701, 2005.

Anirban Banerjee and Jürgen Jost. On the spectrum of the normalized graph Laplacian. *Linear algebra and its applications*, 428(11-12):3015–3022, 2008.

Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep networks? In *Advances in Neural Information Processing Systems*, pages 4261–4271, 2018.

Alessio Benavoli, Alessandro Facchini, and Marco Zaffalon. Quantum mechanics: The Bayesian theory generalized to the space of hermitian matrices. *Phys. Rev. A*, 94:042106, Oct 2016. doi: 10.1103/PhysRevA.94.042106. URL https://link.aps.org/doi/10.1103/PhysRevA.94.042106.

Florent Benaych-Georges and Raj Rao Nadakuditi. The eigenvalues and eigenvectors of finite, low rank perturbations of large random matrices. *Advances in Mathematics*, 227(1):494–521, 2011.

Jakob Bernoulli. *Ars conjectandi*. Impensis Thurnisiorum, fratrum, 1713.

Leonard Berrada, Andrew Zisserman, and Pawan Kumar. Deep Frank-Wolfe for neural network optimization. *arXiv preprint arXiv:1811.07591*, 2018.

N. L. Biggs, E.K. Lloyd, and R. J. Wilson. *Graph theory 1736-1936*. Oxford University Press, 1976.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Alex Bloemendal, Antti Knowles, Horng-Tzer Yau, and Jun Yin. On the principal components of sample covariance matrices. *Probability theory and related fields*, 164(1-2):459–552, 2016.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2009.

Joël Bun, Romain Allez, Jean-Philippe Bouchaud, Marc Potters, et al. Rotational invariant estimator for general noisy matrices. *IEEE Trans. Information Theory*, 62(12):7475–7490, 2016.

Joël Bun, Jean-Philippe Bouchaud, and Marc Potters. Cleaning large correlation matrices: tools from random matrix theory. *Physics Reports*, 666: 1–109, 2017.

A Caticha. Entropic inference and the foundations of physics (monograph commissioned by the 11th brazilian meeting on Bayesian statistics–ebeb-2012, 2012.

Ariel Caticha. Relative entropy and inductive inference. *AIP Conference Proceedings*, 2004. doi: 10.1063/1.1751358.

Ariel Caticha and Adom Giffin. Updating probabilities. *AIP Conference Proceedings*, 2006. doi: 10.1063/1.2423258.

David Chandler. *Introduction to Modern Statistical Mechanics*. Oxford University Press, 1987. ISBN 0195042778.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo

Zecchina. Entropy-SGD: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.

Jinghui Chen and Quanquan Gu. Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint arXiv:1806.06763*, 2018.

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.

Francois Chollet. Keras. https://github.com/fchollet/keras, 2015.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015a.

Anna Choromanska, Yann LeCun, and Gérard Ben Arous. Open problem: The landscape of the loss surfaces of multilayer networks. In *Conference on Learning Theory*, pages 1756–1760, 2015b.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the Cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Fan RK Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

David Cohen-Steiner, Weihao Kong, Christian Sohler, and Gregory Valiant. Approximating the spectrum of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1263–1271, 2018.

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

R. T. Cox. Probability, frequency and reasonable expectation. *American Journal of Physics*, 14(1):1–13, 1946. doi: 10.1119/1.1990764.

Harald Cramer. Mathematical methods of statistics (pms-9). 1946. doi: 10.1515/9781400883868.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical data augmentation with no separate search. *arXiv preprint arXiv:1909.13719*, 2019.

Felix Dangel, Frederik Kunstner, and Philipp Hennig. Backpack: Packing more into backprop. *arXiv preprint arXiv:1912.10985*, 2019.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

Bruno De Finetti. Teoria delle probabilita. einaudi, turin, 1970. *English translation:[51]*, 1974.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

John C Duchi. Introductory lectures on stochastic optimization. *The Mathematics of Data*, 25:99, 2018.

Noureddine El Karoui et al. Tracy-Widom limit for the largest eigenvalue of a large class of complex sample covariance matrices. *The Annals of Probability*, 35(2):663–714, 2007.

Benjamin Eva. Principles of indifference. *The Journal of Philosophy*, 116(7): 390–411, 2019.

Richard Everson and Stephen Roberts. Inferring the eigenvalues of covariance matrices from limited, noisy data. *IEEE transactions on signal processing*, 48(7):2083–2091, 2000.

Illés J Farkas, Imre Derényi, Albert-László Barabási, and Tamas Vicsek. Spectra of real-world graphs: Beyond the semicircle law. *Physical Review E*, 64(2):026704, 2001.

Adina Roxana Feier. *Methods of proof in random matrix theory.* PhD thesis, Harvard University, 2012.

Richard P Feynman, Robert B Leighton, and Matthew Sands. The Feynman lectures on physics; vol. i. *American Journal of Physics*, 33(9):750–752, 1965.

Jack Fitzsimons, Diego Granziol, Kurt Cutajar, Michael Osborne, Maurizio Filippone, and Stephen Roberts. Entropic trace estimates for log determinants, 2017.

Gary William Flake, Steve Lawrence, and C Lee Giles. Efficient identification of web communities. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 150–160. ACM, 2000.

Stanislav Fort, Paweł Krzysztof Nowak, Stanislaw Jastrzebski, and Srini Narayanan. Stiffness: A new perspective on generalization in neural networks. *arXiv preprint arXiv:1901.09491*, 2019.

Zoltán Füredi and János Komlós. The eigenvalues of random symmetric matrices. *Combinatorica*, 1(3):233–241, 1981.

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586, 2018.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a Kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems*, pages 9550–9560, 2018.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: continual prediction with LSTM. 1999.

Semyon Aranovich Gershgorin. Über die abgrenzung der Eigenwerte einer Matrix. *Izvestija Akademii Nauk SSSR, Serija Matematika*, (6):749–754, 1931.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via Hessian eigenvalue density. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2232–2241. PMLR, 2019.

Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W Mahoney, and Joseph Gonzalez. On the computational inefficiency of large batch sizes for stochastic gradient descent. *arXiv preprint arXiv:1811.12941*, 2018.

Gene H Golub and Gérard Meurant. Matrices, moments and quadrature. *Pitman Research Notes in Mathematics Series*, pages 105–105, 1994.

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2012.

Friedrich Götze, A Naumov, and A Tikhomirov. Semicircle law for a class of random matrices with dependent entries. *arXiv preprint arXiv:1211.0389*, 2012.

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.

Diego Granziol and Stephen Roberts. An information and field theoretic approach to the grand canonical ensemble, 2017.

Diego Granziol, Binxin Ru, Stefan Zohren, Xiaowen Dong, Michael Osborne, and Stephen Roberts. Entropic spectral learning for large-scale graphs. *arXiv preprint arXiv:1804.06802*, 2018.

Diego Granziol, Binxin Ru, Stefan Zohren, Xiaowen Dong, Michael Osborne, and Stephen Roberts. Meme: An accurate maximum entropy method for efficient approximations in large-scale machine learning. *Entropy*, 21(6): 551, 2019a.

Diego Granziol, Xingchen Wan, Timur Garipov, Dmitry Vetrov, and Stephen Roberts. MLRG deep curvature. *arXiv preprint arXiv:1912.09656*, 2019b.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Walid Hachem, Philippe Loubaton, Jamal Najim, and Pascal Vallet. On bilinear forms based on the resolvent of large random matrices. In *Annales de l'IHP Probabilités et statistiques*, volume 49, pages 36–63, 2013.

Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using Networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

Insu Han, Dmitry Malioutov, and Jinwoo Shin. Large-scale log-determinant computation through stochastic Chebyshev expansions. In *International Conference on Machine Learning*, pages 908–917, 2015.

Felix Hausdorff. Summations methoden und moment folgen. i. *Mathematische Zeitschrift*, 9(1-2):74–109, 1921.

Hangfeng He and Weijie J Su. The local elasticity of neural networks. *arXiv preprint arXiv:1910.06943*, 2019.

Haowei He, Gao Huang, and Yang Yuan. Asymmetric valleys: Beyond sharp and flat local minima. *arXiv preprint arXiv:1902.00744*, 2019.

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, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

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.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, pages 2160–2170, 2018.

Giora Hon. The art of conjecturing, together with letter to a friend on sets in court tennis - by Jacob Bernoulli (trans. with an introduction and notes by edith dudley sylla). *Centaurus*, 50(4):335â337, 2008. doi: 10.1111/j.1600-0498.2008.00117.x.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

M.F. Hutchinson. A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990a.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Prateek Jain, Praneeth Netrapalli, Sham M Kakade, Rahul Kidambi, and Aaron Sidford. Parallelizing stochastic gradient descent for least squares regression: mini-batching, averaging, and model misspecification. *The Journal of Machine Learning Research*, 18(1):8258–8299, 2017.

Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, Jacek Tabor, Kyunghyun Cho, and Krzysztof Geras. The break-even point on the optimization trajectories of deep neural networks. In *International Conference on Learning Representations*, 2020.

Stanisław Jastrzkbski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. Three factors influencing minima in SGD. *arXiv preprint arXiv:1711.04623*, 2017.

Stanisław Jastrzkbski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. On the relation between the sharpest directions of DNN loss and the SGD step length. 2018.

E. T. Jaynes. Information theory and statistical mechanics. *Phys. Rev.*, 106: 620–630, May 1957a.

E. T. Jaynes. Information theory and statistical mechanics. *Phys. Rev.*, 106:620–630, May 1957b. doi: 10.1103/PhysRev.106.620. URL http://link.aps.org/doi/10.1103/PhysRev.106.620.

Edwin T Jaynes. Concentration of distributions at entropy maxima. *ET Jaynes: Papers on probability, statistics, and statistical physics*, pages 315–336, 1983.

James M Joyce. Bayesianism. 2004.

Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in neural information processing systems*, pages 586–594, 2016.

Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from Adam to SGD. *arXiv preprint arXiv:1712.07628*, 2017.

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. *arXiv preprint arXiv:1609.04836*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Ming Zhou, Klaus Neymeyr, and Thomas Hofmann. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. *arXiv preprint arXiv:1805.10694*, 2018.

Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

Harold Kushner and G George Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.

L. D. Landau, Lifshitz E. M., and PitaevskiiÌ L. P. *Statistical physics*. Elsevier, 2012.

Yann LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007.

Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Advances in Neural Information Processing Systems*, pages 11669–11680, 2019.

Lin Lin, Yousef Saad, and Chao Yang. Approximating spectral densities of large matrices. *SIAM Review*, 58(1):34–65, 2016.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR) 2017 Conference Track*, April 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

David JC MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, pages 13132–13143, 2019.

Vladimir Alexandrovich Marchenko and Leonid Andreevich Pastur. Distribution of eigenvalues for some sets of random matrices. *Matematicheskii Sbornik*, 114(4):507–536, 1967.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. 1993.

James Martens. Deep learning via Hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

James Martens. *Second-order optimization for neural networks*. PhD thesis, University of Toronto, 2016.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

James Martens and Ilya Sutskever. Training deep and recurrent networks with Hessian-free optimization. In *Neural networks: Tricks of the trade*, pages 479–535. Springer, 2012.

Mark D McDonnell. Training wide residual networks for deployment using a single bit for each weight. *arXiv preprint arXiv:1802.08530*, 2018.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. *arXiv preprint arXiv:1708.02182*, 2017.

N David Mermin. Stirling's formula! *American Journal of Physics*, 52(4): 362–365, 1984.

Gérard Meurant and Zdeněk Strakoš. The Lanczos and conjugate gradient algorithms in finite precision arithmetic. *Acta Numerica*, 15:471–542, 2006.

Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42. ACM, 2007.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006a.

Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006b.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

Sean O'Rourke et al. A note on the Marchenko-Pastur law for a class of random matrices with dependent entries. *Electronic Communications in Probability*, 17, 2012.

Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *nature*, 435(7043):814, 2005.

Vardan Papyan. The full spectrum of deepnet Hessians at scale: Dynamics with SGD training and sample size. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5012–5021. PMLR, 2019.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in Pytorch. 2017.

Barak A Pearlmutter. Fast exact multiplication by the Hessian. *Neural computation*, 6(1):147–160, 1994.

Jeffrey Pennington and Yasaman Bahri. Geometry of neural network loss surfaces via random matrix theory. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2798–2806. JMLR. org, 2017.

Jeffrey Pennington and Pratik Worah. The spectrum of the Fisher information matrix of a single-hidden-layer neural network. In *Advances in Neural Information Processing Systems*, pages 5410–5419, 2018.

Steven J Phillips, Robert P Anderson, Miroslav Dudík, Robert E Schapire, and Mary E Blair. Opening the black box: An open-source release of Maxent. *Ecography*, 40(7):887–893, 2017.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4): 838–855, 1992.

Steve Press, Kingshuk Ghosh, Julian Lee, and Ken A. Dill. Nonadditive entropies yield probability distributions with biases not warranted by the data. *Physical Review Letters*, 111(18), Jan 2013. doi: 10.1103/physrevlett.111.180604.

Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Farbod Roosta-Khorasani and Michael W Mahoney. Sub-sampled newton methods ii: Local convergence rates. *arXiv preprint arXiv:1601.04738*, 2016.

R. D. Rosenkrantz. Confidence intervals vs Bayesian intervals (1976). *E. T. Jaynes: Papers on Probability, Statistics and Statistical Physics*, page 149â209, 1989a. doi: 10.1007/978-94-009-6581-2_9.

R. D. Rosenkrantz. *Where Do we stand on maximum entropy? (1978)*, pages 210–314. Springer Netherlands, Dordrecht, 1989b. ISBN 978-94-009-6581-2. doi: 10.1007/978-94-009-6581-2_10.

Nicolas L Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *Advances in neural information processing systems*, pages 849–856, 2008.

Binxin Ru, Mark McLeod, Diego Granziol, and Michael A Osborne. Fast information-theoretic Bayesian optimisation. *arXiv preprint arXiv:1711.00673*, 2017.

Tony Saad and Giovanna Ruai. Pymaxent: A Python software for Maximum Entropy moment reconstruction. *SoftwareX*, 10:100353, 2019.

Levent Sagun, Léon Bottou, and Yann LeCun. Eigenvalues of the Hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Levent Sagun, Utku Evci, V. Ugur Güney, Yann N. Dauphin, and Léon Bottou. Empirical analysis of the Hessian of over-parametrized neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

S. M. Samuels and William Feller. An introduction to probability theory and its applications. *Technometrics*, 15(2):420, 1973. doi: 10.2307/1267002.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

Robert E Schapire. The strength of weak learnability. *Machine learning*, 5 (2):197–227, 1990.

Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

John Skilling. The eigenvalues of mega-dimensional matrices. In *Maximum Entropy and Bayesian Methods*, pages 455–466. Springer, 1989.

C. Ray Smith and Gary J. Erickson. Probability theory and the associativity

equation. *Maximum Entropy and Bayesian Methods*, page 17â30, 1990. doi: 10.1007/978-94-009-0683-9_2.

Charles Stein. Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. Technical report, Stanford University Stanford United States, 1956.

Charles Stein. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, Volume 2: Probability Theory*, pages 583–602, Berkeley, Calif., 1972. University of California Press.

Gabor Szeg. *Orthogonal polynomials*, volume 23. American Mathematical Soc., 1939.

Daniel Yasumasa Takahashi, Joao Ricardo Sato, Carlos Eduardo Ferreira, and André Fujita. Discriminating different classes of biological networks by analyzing the graphs spectra distribution. *PLoS One*, 7(12):e49949, 2012.

Terence Tao. *Topics in random matrix theory*, volume 132. American Mathematical Soc., 2012.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Magnus Tornstad. Evaluating the practicality of using a Kronecker-factored approximate curvature matrix in Newton's method for optimization in neural networks, 2020.

Craig A Tracy and Harold Widom. Level-spacing distributions and the Airy kernel. *Communications in Mathematical Physics*, 159(1):151–174, 1994.

Phuong Thi Tran et al. On the convergence proof of AMSGrad and a new version. *IEEE Access*, 7:61706–61716, 2019.

M. Tribus. Preface. *Rational Descriptions, Decisions and Designs*, pages xv–xviii, 1969. doi: 10.1016/b978-0-08-006393-5.50005-2.

Shubhendu Trivedi and Risi Kondor. Cmsc 35246: Deep learning. lecture 6: Optimization for deep neural networks. 2017.

Shashanka Ubaru and Yousef Saad. Applications of trace estimation techniques.

Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast Estimation of tr (f (a)) via Stochastic Lanczos Quadrature. 2016.

Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of tr(f(a)) via stochastic Lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099, 2017.

Edwin R Van Dam and Willem H Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its applications*, 373:241–272, 2003.

Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

Oriol Vinyals and Daniel Povey. Krylov subspace descent for deep learning. In *Artificial Intelligence and Statistics*, pages 1261–1268, 2012.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Peter Walley. Statistical reasoning with imprecise probabilities. 1991. doi: 10.1007/978-1-4899-3472-7.

Alexander Weiße, Gerhard Wellein, Andreas Alvermann, and Holger Fehske. The kernel polynomial method. *Reviews of modern physics*, 78(1):275, 2006.

Hermann Weyl. Das asymptotische verteilungsgesetz der eigenwerte linearer partieller differentialgleichungen (mit einer anwendung auf die theorie der hohlraumstrahlung). *Mathematische Annalen*, 71(4):441–479, 1912.

Eugene P Wigner. Characteristic vectors of bordered matrices with infinite dimensions i. In *The Collected Works of Eugene Paul Wigner*, pages 524–540. Springer, 1993.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

Lei Wu, Zhanxing Zhu, et al. Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv preprint arXiv:1706.10239*, 2017.

Lei Wu, Chao Ma, and E Weinan. How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective. In *Advances in Neural Information Processing Systems*, pages 8279–8288, 2018.

Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Self-training with noisy student improves ImageNet classification, 2019.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. In *Advances in Neural Information Processing Systems*, pages 4949–4959, 2018.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. PyHessian: Neural networks through the lens of the Hessian. *CoRR*, abs/1912.07145, 2019.

Hongwei Yong, Jianqiang Huang, Xiansheng Hua, and Lei Zhang. Gradient centralization: A new optimization technique for deep neural networks. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, volume 12346 of *Lecture Notes in Computer Science*, pages 635–652. Springer, 2020.

Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 6022–6031. IEEE, 2019.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. *arXiv preprint arXiv:1810.12281*, 2018.

Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. In *Advances in Neural Information Processing Systems*, pages 8194–8205, 2019a.

Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, pages 9593–9604, 2019b.

Dongruo Zhou, Yiqi Tang, Ziyan Yang, Yuan Cao, and Quanquan Gu. On the convergence of adaptive gradient methods for nonconvex optimization. *CoRR*, abs/1808.05671, 2018.

Huaiyu Zhu, Christopher KI Williams, Richard Rohwer, and Michal Morciniec. Gaussian regression and optimal finite dimensional linear models. 1997a.

Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural computation*, 9 (8):1627–1660, 1997b.

A | Method of Relative Entropy

Consider devising a general method allowing us to update from a prior distribution $q(x)$ to a posterior $p(x)$ under the input of new information. We create a transitive ranking system by assigning a real number to each probability distribution, i.e. if we prefer p_1 to p_2 and we prefer p_2 to p_3 all relative to our prior q. then we have $\mathcal{S}(p_1, q) > \mathcal{S}(p_2, q) > \mathcal{S}(p_3, q)$. This explains why entropies are real numbers that need to be maximised. Anything that can be ranked can be mapped to the real numbers and we wish to have the highest ranking. It can be proven [Caticha, 2004, Caticha and Giffin, 2006] that restrictions to the functional form exist under the following conditions:

Axiom 1. Locality *Local information has local effects*

When new information does not refer to the domain D of the variable x the conditional probabilities $p(x|D)$ remain unaltered. Consequently non-overlapping domains contribute additively and we can represent the entropic functional as an integral, namely

$$\mathcal{S}[p, q] = \int_{x \in \mathcal{D}} dx\, F(p(x), q(x), x) \tag{A.1}$$

where F is unknown.

Axiom 2. Coordinate invariance: *ranking cannot depend on the coordinate system*

$$\mathcal{S}[p] = \int dx\, m(x) \Phi(\frac{p(x)}{m(x)}, \frac{q(x)}{m(x)}) \tag{A.2}$$

where $\Phi(\alpha, \beta, m, x) \equiv \frac{1}{m} F(\alpha m, \beta m, x)$. and $m(x)$ is a probability density, which implies that $q(x)/m(x)$ and $p(x)/m(x)$ are coordinate invariant.

Axiom 3. system independence *For a system of independent systems the inference procedure should not be dependent on whether they are treated*

separately or jointly. We are thus restricted us to a family of distributions characterized by $\eta \in \mathbb{R}_+$,

$$\mathcal{S}[p,q] = \frac{1}{\eta(\eta+1)}\left(1 - \int dx\, p(x)^{\eta+1} q(x)^{-\eta}\right) \tag{A.3}$$

which are known as Tsallis or Renyi entropies. Under the further assumption that two parties with the same prior and same information must make the same inference we conclude that only a singular value of η is permitted. Given the experimental success of $\eta = 0$ to predict Bayes rule along with classical, quantum and statistical mechanics we consider the functional

$$\mathcal{S}[p,q] = -\int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \tag{A.4}$$

The idea that non Shannon entropies introduce inherent correlations/dependence is known [Press et al., 2013]. We further note that for a constant $q(x)$, that the entropic functional becomes

$$S[p] = -\int p(x) \log p(x) dx + \int p(x) \log q(x) dx \tag{A.5}$$

Where the second term does not affect the optimisation and so can be dropped.

A.0.1 Bayes' Rule from the Method of Relative Entropy

In this section, we show that Bayes' rule can be directly inferred from *the method of relative entropy*, under the principle of minimal update. Bayes rule is often quoted as a restatement of the product rule.

$$q(\theta|x') = q(\theta)\frac{q(x'|\theta)}{q(x')} \tag{A.6}$$

Where δ is the dirac delta function, θ are the model parameters and x is a data point. This formula is not a posterior, but a prior conditional probability. We invoke the principle of minimal update, which stipulates that the web of beliefs need only be revised to the extent required by new data.

A.0.1.1 Principle of Minimal Update (PMU)

All distributions of the form

$$p(\theta, x') = \delta(x - x')p(\theta|x') \tag{A.7}$$

fit the data, we invoke the PMU by saying no further revision is needed and that $p(\theta|x') = q(\theta|x')$ thus

$$p(\theta) = \int dx\delta(x - x')q(\theta|x') = q(\theta|x') = q(\theta)\frac{q(x'|\theta)}{q(x')} \tag{A.8}$$

This differs from the standard Bayesian update as it adds in the requirement that the posterior probability distribution satisfies the constraints. It also does so in a maximally configurable (most entropic) way.

$$\begin{aligned}
&\delta\Bigg(- \int dxd\theta p(x,\theta)\left[\log\frac{p(x,\theta)}{q(x,\theta)}\right] - \alpha\int[dxd\theta p(x,\theta) - 1] \\
&- \int dx\lambda(x)\left[d\theta p(x,\theta) - \delta(x - x')\right] - \beta\left[\int d\theta p(x,\theta)f(\theta) - < F >\right] = 0\Bigg) \\
&\xrightarrow{thus} P(x,\theta) = q(x,\theta)\frac{\exp\left(-\lambda(x)\right)\exp\left(-\beta f(\theta)\right)}{z}
\end{aligned} \tag{A.9}$$

We can make progress by noting that the distribution must satisfy the data constraints $\int d\theta p(x,\theta) = \delta(x - x')$. This is basically enforcing the model to

be compatible with the data but using the language of constraints.

$$P(\theta) = \frac{\int dx q(\theta|x)\delta(x - x') \exp\left(-\beta f(\theta)\right)}{\int dx d\theta q(\theta|x) \exp\left(-\beta f(\theta)\right)} = q(\theta|x') \frac{\exp\left(-\beta f(\theta)\right)}{\int dx d\theta q(\theta|x) \exp\left(-\beta f(\theta)\right)} \tag{A.10}$$

This reduces to Bayes rule when $\beta = 0$. Which means there is no penalty for violating the mean value constraints, or alternatively when there are no mean value constraints.