Identifying and Exploiting Structures for Reliable Deep Learning

Amartya Sanyal

St Hugh's College

A thesis submitted for the degree of
Doctor of Philosophy

University of Oxford
Trinity 2021.

to the Weasleys (even Percy)
and Professor Shonku

Acknowledgements

My D.Phil would have been far less enjoyable without the many meetings, supervision and otherwise, with my D.Phil advisors Varun Kanade and Philip Torr. I am grateful to them for allowing me to pursue research questions of my own choosing and putting up with me despite most of them failing. Varun's ability to constantly see through the inconsequential details and isolate the fundamental questions has been a constant surprise for me and something I hope I have learnt from. I also thank Varun for all I have learnt from him about computational learning theory and the many interesting chats about machine learning which I believe has had a lasting impact on what questions I find interesting. I thank Phil for constantly believing in me, spreading his ever-present extremely infectious energy, eagerness to dive into new research topics, seeing the bigger picture whenever I have proposed a new idea, and most importantly always having time for me despite being a very busy person. I will lie if I do not admit that I have always believed that he has a doppelganger at his service. Combined together, I have had an unique opportunity to work in both the theory and practice of machine learning.

Without the constant support and determination of my parents, this thesis (and most of my education) would have been impossible. For that and more, I am forever indebted to them. I also owe it to everyone in my childhood including my extended family, who has contributed to keep me interested in the mathematical sciences. I should specifically thank Purushottam, Prateek, and all my teachers at IITK who introduced me to machine learning in a way that I decided to pursue it for my DPhil. I also thank my friends from IIT — Bhuvesh, Asim, Nirbhay, Nihar, Sivasankar, and many more.

I thank Prof. Shimon Whiteson and Prof. Yarin Gal for examining my transfer and my confirmation, respectively. I thank Prof. Pawan Kumar for taking the time to be on my transfer, confirmation as well as my final thesis examination committee. Their advice has been invaluable for guiding me at the respective stages of my D.Phil. I also thank Prof. Pranjal Awasthi for agreeing to be on my thesis committee.

I owe a special thanks to Puneet for indulging all my crazy ideas over the last three years, re-reading and re-writing my drafts, and being my longest and closest research collaborator. I have been very lucky to meet a number of extraordinary researchers and collaborators on the way including Adrià Gascón, Matt Kusner, Lorenzo Rosasco, Patrick Rebeschini, Edward Grefenstette, Tim Rocktäschel, Stuart Golodetz, Tomas Vaškevičius, and Mario Lezcano Casado. I hope I have learned a lot from each of them.

I thank David for keeping me constantly informed and interested in optimisation. I have always enjoyed discussing with him our innumerable ideas. I thank Limor for introducing me to causality and interpretability. Working on a paper with David

and Limor is on the very top of my todo list. TVG has been a wonderful group to be a part of especially to be able work so closely with Jishnu, Viveka, Tom, Pau, Harirat, and Yuge (in chronological order). I have learnt a lot from each of them and the diversity in their research expertise has made this a very enriching experience.

St. Hugh's college (along with its vast gardens) and the St. Hugh's MCR has been the most wonderful place to live in and be a part of, especially during the pandemic. I should specifically thank Alex for our many escapades and for always saying a joke when I needed to hear it (or not), Lizzie for being the best neighbour and for pointing out how my cooked dishes are consistently brown, Josh T for many things including the timely car rides to the McDonald's and KFC's, Kat for being a constant reminder of the amazing powers of a cup of coffee, Anwar, Florence, and Ed for being excellent reasons to go to pubs, socials, and formals, Josh C for sharing my love for Bourdain, Sharan for his unlimited energy and positivity, which often (luckily) overflowed into me, Lili for making dinners a social and delicious affair, Qian and Brandon for making my time at Oxford worth remembering, Michelle and Yves for being the best running partners, and Lukas and Arnaud for being the constant friends that they have been.

I also thank the Department of Computer Science at the University of Oxford for providing me with a productive research atmosphere and workspace, as well as the department of Engineering Science for allowing me to be their member. I cannot thank Julie and Sarah enough for making all the admin work seem like a breeze.

This work was made possible by the generous support of the Turing doctoral studentship from The Alan Turing Institute. I thank Sam, Georgia, Ben, and everyone else at the Turing institute for making sure I had all the help I needed. I also thank my colleagues at the Turing institute, especially the Oxford cohort — Florentine, Sanna, Prateek, Julien, Michael, Jessie, Francesco, Charlie, and Ayman.

Last but not the least I also want to thank everyone who made my life easier during the COVID-19 pandemic — everyone at Brew, Nine2Nine, RnC, Vinny's Cafe, Pepper's, Ed Reid and everyone in Hugh's gardening team, the college porters and many more. Finally, I also thank Bourdain, Hemingway and Salinger for taking me away from my work when I most needed a break.

Contents

Abstract

Deep learning research has recently witnessed an impressively fast-paced progress in a wide range of tasks including computer vision, natural language processing, and reinforcement learning. The extraordinary performance of these systems often gives the impression that they can be used to revolutionise our lives for the better. However, as recent works point out, these systems suffer from several issues that make them unreliable for use in the real world, including vulnerability to adversarial attacks (Szegedy et al. [248]), tendency to memorise noise (Zhang et al. [292]), being over-confident on incorrect predictions (miscalibration) (Guo et al. [99]), and unsuitability for handling private data (Gilad-Bachrach et al. [88]). In this thesis, we look at each of these issues in detail, investigate their causes, and propose computationally cheap algorithms for mitigating them in practice.

To do this, we identify structures in deep neural networks that can be exploited to mitigate the above causes of unreliability of deep learning algorithms. In Chapter 4, we show that minimizing a property of matrices, called stable rank, for individual weight matrix in a neural network reduces the tendency of the network to memorise noise without sacrificing its performance on noiseless data.

In Chapter 5, we prove that memorising label noise or doing improper representation learning makes achieving adversarial robustness impossible. Chapter 6 shows that a low-rank prior on the representation space of neural networks increases the robustness of neural networks to adversarial perturbations without inducing any tradeoff with accuracy in practice.

In Chapter 7, we highlight the use of focal loss, which weights loss components from individual samples differentially by how well the neural network classifies each of them, as an alternative loss function to cross-entropy for minimizing miscalibration in neural networks.

In Chapter 8, we first define a new framework called *Encrypted Prediction As A Service* (EPAAS) along with a set of computational and privacy constraints. Then we propose the use of a Fully Homomorphic Encryption [84] scheme which can be used with a Binary neural network [61], along with a set of algebraic and computational tricks, to satisfy all our conditions for EPAAS while being computationally efficient.

Bibliographic Notes

The material in Chapter 4 is based on Sanyal et al. [232], which is published as a Spotlight paper in International Conference on Learning Representations (ICLR), 2020 and is co-authored with Dr. Puneet Dokania and my DPhil advisor Prof. Philip H.S. Torr. The material in Chapter 5 is primarily based on Sanyal et al. [233], which is published as a Spotlight paper in the International Conference on Learning Representations (ICLR) 2021. Chapter 6 is based on the work in Sanyal et al. [230]. Both of these works are co-authored with Dr. Puneet Dokania and my D.Phil advisors Prof. Varun Kanade and Prof. Philip H.S. Torr. The material in Chapter 7 is mostly based on the work in Mukhoti et al. [188], which is published in the Advances in Neural Information Processing Systems (NeurIPS) 2020 and was also presented as a Spotlight paper in the Workshop on Uncertainty and Robustness in Deep Learning, held at the International Conference in Machine Learning (ICML), 2020. The material in Chapter 8 is based on Sanyal et al. [231], which was published in the International Conference in Machine Learning (ICML), 2018 and is co-authored with Prof. Matt Kusner, Dr. Adrià Gascón and Prof. Varun Kanade. I have also co-authored de Jorge et al. [63], which is published in the International Conference on Learning Representations (ICLR) 2021 but is not included in this thesis.

Chapters 2 and 3 are written solely by me but the original research is not performed by me. The original sources of the material is cited in the relevant parts and my contribution is in unifying it and presenting it in the context of the original material presented in this thesis. I am the primary author and the sole student author of the works presented in Chapters 4 to 6. The primary student authors of the work presented in Chapter 7 is Jishnu Mukhoti and Viveka Kulharia. While, they have done most of the experimental work, I have worked on the theoretical and conceptual part of this work. I am also the sole student author of the work presented in Chapter 8.

List of Theorems

List of Figures

List of Tables

List of Algorithms

1. Introduction

Machine learning research, and in particular deep learning research, has shown remarkable progress in recent years. On several tasks, machine learning algorithms have already surpassed human-level performance, albeit only on some specific measures of performance. For example, on a popular computer vision dataset *Imagenet*, He et al. [109] obtained a (top-5) classification error of 4.9% in 2015, surpassing the human benchmark of 5.1% (reported in Russakovsky et al. [225]). Since then, there have been consistent improvements in classification accuracies on the dataset from an error of 4.7% in 2017 [51], to 3.4% in 2019 [216], and as low as 1.2% in 2020 [210]. It is not just computer vision that has enjoyed such fast progress but also tasks like language modelling [95, 281, 268], language translation [170, 278, 82], and medical image segmentation [220, 78, 254].

The outcomes of these studies have not remained within the confines of research labs; they are now actively deployed in multiple sectors in the real world by governments and private companies alike. Automobile companies, including Tesla [6], Waymo [9], GMCruise [2], and ArgoAI [1], are actively deploying deep neural networks in their autonomous driving systems. Companies like Pixsy [4], Tineye [8], and Salesforce [5] are among those who are using computer vision algorithms to search for images in large databases rapidly. In health care, companies like Infervision, Deepmind, and Microsoft Research are using deep learning algorithms to speed up, other-wise time-consuming, manual activities like detecting cancerous tissues in medical images [2][3] or segmenting medical scans during radiotherapy preparations [4].

Online services like Tilde [7], Bing [1], and Lengoo [3] uses deep neural networks to offer language translation services. A prominent example of the use of deep learning by governments is the *EU Presidency Translator* [5], developed by Tilde and

[1]https://www.theverge.com/2017/8/16/16155254/argo-ai-ford-self-driving-car-autonomous

[2]https://techcrunch.com/2018/11/30/infervision-medical-imaging-280-hospitals/

[3]http://www.imperial.ac.uk/news/183293/research-collaboration-aims-improve-breast-cancer/

[4]https://www.cambridgeindependent.co.uk/news/addenbrooke-s-and-microsoft-research-create-world-first-ai-technology-to-speed-up-radiography-preparations-9147027/

[5]https://www.eu2020.de/eu2020-en/presidency/uebersetzungstool/2361002

used by Finland during its 2019 presidency of the EU council to allow delegates, and attendees to overcome language barriers and access multilingual information by automatically translating documents and local websites. Perhaps playing a more direct role in shaping our daily lives and world view are social media companies like Twitter and Facebook, who use deep learning algorithms to recommend social content and connections.

This surge in deployment of deep learning algorithms is primarily due to its remarkable feats in performance-based metrics like the accuracy of prediction in classification tasks, *Intersection-Over-Union (IoU)* in image segmentation tasks, *BLEU score* in translation tasks, and *perplexity* in language modelling tasks. In turn, research in deep learning has also focused primarily on improving these performance-based metrics. A large portion of these improvements has come from designing novel regularisation methods and exploiting new inductive biases in deep learning algorithms. For example, regularisation techniques like Batch Normalization [126], DropOut [243], Label Smoothing [249], and DropConnect [176] provided state-of-the-art accuracy in their respective tasks at the time of their development. Large performance increases have also come from incorporating the right inductive biases in learning algorithms. In deep learning, inductive biases have mainly been incorporated in the form of changes to the architecture of the neural network. Convolutional Layers [156] and Recurrent Layers [96] provide domain-dependent inductive biases by introducing translation invariance and temporal invariance, respectively. The positional embeddings in transformer architectures [266] allow the model to encode absolute and relative positions and positionally invariant relationships. In language models, using Byte Pair Encoding [236, 263] for words reflects the inductive bias that words composed of similar sub-words are related. All of these inductive biases have provided significant advances in the performance of machine learning algorithms. These techniques identify structures that can be helpful for the performance-based metric on the task at hand and then exploit it through a specially designed regulariser, data-augmentation, or architectural change.

1.1 Challenges with the reliability of deep learning

However, as recent works have pointed out, state-of-art deep neural networks, while providing impressive performance, suffer from a range of issues that hurt their reliability:

- **Generalisation**: Zhang et al. [292] observed that neural networks, that achieve small test errors, are also equally capable of memorising a random labelling of the observed data. This is troubling as in deep learning, future performance on unseen data is usually estimated from performance on observed data. However, the experiments of Zhang et al. [292] indicate that such trust in a neural network performance on the observed data might be misplaced. This calls for further research into certifying the generalisability of a learnt model on unseen data.

- **Robustness**: Szegedy et al. [248] pointed out that deep neural networks are extremely vulnerable to imperceptible perturbations to input images. They show that, with minimal adversarially crafted distortions to input images, a network's error could be increased from 2.1% to 100% on a commonly used computer vision dataset called MNIST [156]. This vulnerability is referred to as adversarial vulnerability. Subsequent works have shown that this phenomenon is seen across a wide variety of deep learning models and datasets.

- **Calibration**: Deep neural networks, used for classification tasks, do not just output a class label but also a probability distribution over the possible classes. The probability value, associated with a class, is supposed to indicate the likelihood of that class being the correct output. When this property is satisfied by a machine learning model, the machine learning model is said to be calibrated. In 2005, Niculescu-Mizil and Caruana [197] showed that neural networks are well-calibrated compared to other prevalent classifiers at the time. However, in 2017, Guo et al. [99] showed that with increasing focus towards producing highly accurate classifiers, state-of-the-art neural networks are extremely miscalibrated and as a result, are over-confident on their incorrect predictions.

- **Privacy**: Apart from these issues, which relate to the quality of the prediction of a deep neural network, modern deep learning models have also been built without considering the privacy of the user. If these machine learning models are to be deployed in the real world to cater to the needs of a large number of people, the privacy of the users needs to be guaranteed. One such guarantee of privacy could be to allow users to use a machine learning model, operated by a private company, by sending over an encrypted version of their private data to the private company and receiving an encrypted result, computed by the machine learning model. However, this is computationally very expensive with modern deep learning architectures and thus unappealing for private companies who want their systems to have a high throughput, so that they can cater to a large number of requests.

In this thesis, we consider the above-mentioned issues with reliability in deep neural networks, identify metrics to measure them in practice and propose ways to mitigate them. To do this, we investigate specific causes for these vulnerabilities, pinpoint structures in the data and the model architecture, which can be exploited to mitigate the recognised cause, and design easy-to-use algorithms to implement to exploit these structures in practice and verify whether it improves on the metrics of reliability. Our approach is similar to what has been used to boost performance-based metrics of deep neural networks but instead of focusing on metrics like test accuracy, we look at metrics of reliability like avoiding memorisation, increasing adversarial robustness and calibration, and enabling privacy of data. In the next section, we briefly comment on how our contributions to each of these topics are organised in this thesis.

1.2 Summary of contributions and organisation

Chapter 2 defines the main problem of learning from data and discusses the importance of the various components of the learning problem. This chapter provides the necessary background required to understand the rest of the thesis and puts it in the context of the existing scientific research in the field of machine learning. Chapter 3

discusses the issues with the reliability of deep learning methods, which are briefly mentioned in the previous section, in greater detail along with a brief survey of the existing works in this field.

In Chapter 4, we look at the problem of generalisation in deep learning. Inspired by recent theoretical research [196], we identify a structure called stable rank [224], measured for individual weight matrices of a neural network, as being important for the generalisation of deep neural networks. Then we propose an algorithm called Stable Rank Normalization (SRN), with theoretical guarantees, to control the stable rank of neural networks. Finally, we devise principled experimental methods to measure generalisation in practice and show, through experiments on a wide range of datasets and architectures, that SRN indeed improves the generalisation of deep neural networks.

Chapter 5 identifies two specific reasons for the lack of adversarial robustness in machine learning models. In particular, we show theoretically and experimentally how overfitting label noise can give a false sense of security in that it might not increase test error but can drastically increase adversarial error. We identify the second cause to be improper representation learning and show that using incorrect representations can get low test error but can never get small adversarial error whereas using a different representation can achieve both low test and adversarial error simultaneously even without needing more data. Continuing from this, in Chapter 6, we show that a low-rank prior on the representation space of neural networks, if applied properly, can impart better adversarial robustness for deep neural networks. We propose a computationally efficient algorithm that scalably learns a neural network with low rank representations without significant modifications to the architecture. Along with a large boost to adversarial robustness, while maintaining test accuracy, our experiments show that this has implications for compression of the learned representations and the model as well.

We find that minimising the cross-entropy loss function minimises the difference between the softmax distribution and the one-hot encoding of the labels for all samples, irrespective of how well the model classifies individual samples. This leads to

a phenomenon referred to as NLL overfitting [99], which is responsible for miscalibration in deep neural networks. In Chapter 7, we propose the use of an alternative loss function, popularly known as *focal loss* [165], that tackles this by weighting loss components generated from individual samples by how well the model classifies each of them. We show, through experiments on a diverse set of datasets and model architectures, that focal loss is much better than its competitors at producing calibrated models without sacrificing test accuracy.

In Chapter 8, we look at the problem of preserving the privacy of the data while being able to do prediction on it using a machine learning model. To this end, we define a framework called *Encrypted Prediction As A Service (EPAAS)* along with a set of privacy and computational requirements that an EPAAS framework should satisfy. We find that a cryptography protocol called *Fully Homomorphic Encryption (FHE)* [84] while being perfectly suited to satisfy our conditions for EPAAS, is computationally very expensive when applied to deep neural networks. We propose the use of binary neural networks, which along with a set of algebraic and computational tricks, makes the application of FHE on deep neural networks feasible. We show, experimentally, that our approach suffers very little in terms of accuracy while satisfying all the criterion of EPAAS.

Appendix A lists some mathematical preliminaries including inequalities and definitions that are used elsewhere in the thesis and is meant to save time for the reader by not having to look up external resources. Appendix B describes the datasets and neural network architectures, along with details of training algorithms, that have been used in the main chapters of the thesis. These are mostly details a deep learning practitioner would be familiar with but have been provided for the sake of completeness and reproducibility. Appendices C to F contains formal proofs and additional figures and tables for the material presented in Chapters 4 to 7 respectively.

Notations All vectors are denoted with lower case bold $\mathbf{v}, \mathbf{x}$ letters and all matrices are represented by upper case bold $\mathbf{A}, \mathbf{B}$ letters. Some commonly used vectors are $\mathbf{x}_i$ to represent the i^{th} example in the dataset and, $\mathbf{z}_l$ and $\mathbf{a}_l$ to represent the

pre-activation and post-activation vectors of the l^{th} layer in a neural network, respectively. Commonly used matrices include $\mathbf{W}_l$ to represent the weight matrix of the l^{th} layer and $\mathbf{A}_l$ to represent the matrix of activations for the i^{th} layer on the entire dataset. A collection of objects is represented by upper case scripted $\mathcal{X}$, $\mathcal{Y}$ letters. An example of a collection is a set or a vector space. When we define a random variable to take a value from a set, with a slight abuse of notation, we will also denote that random variable with the same notation.

Any lemmas or theorems that are borrowed from existing literature are numbered alphabetically (eg. Lemma A and Theorem A) and any result that is original to this thesis are numbered numerically (eg. Lemma 1 and Theorem 1). Unless otherwise stated, d represents the dimensionality of the input domain of a machine learning model, k represents the number of classes in a multi-class classification problem, η represents the uniform label noise rate, L represents the depth of a neural network and W represents the width i.e. the maximum width of any of its layers. The rest of the notations that are specific to each chapter are defined within the chapter itself.

2. Preliminaries of Learning Theory

2.1 The Learning from Data Problem

This chapter will first discuss the various components of the *Learning from data* problem and then formally state the learning problem. Let $\mathcal{X}$ and $\mathcal{Y}$ be two random variables defined over the instance space and the label space, respectively. With a slight abuse of notation, we use the same notations $\mathcal{X}$ and $\mathcal{Y}$ to refer to the instance space and the label space, as well. For most problems in this thesis, $\mathcal{X}$ will be a compact domain in $\mathbb{R}^d$ where d is the dimension of the problem and $\mathcal{Y}$ will be an unordered finite set. In an image classification problem, the space of vector representations of natural images is an example of the instance space and the set of one-hot representations of the classes in a multi-class classification problem is an example of the label space. A distribution $\mathcal{D}$, commonly referred to as the *data distribution*, is defined on the product space $\mathcal{X} \times \mathcal{Y}$. Importantly, this distribution is fixed but unknown to the learner. A finite sample $\mathcal{S}_m = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, referred to as a *training dataset*, is created by sampling m independently and identically distributed (i.i.d.) points from $\mathcal{D}$.

The next component of the learning problem is the *hypothesis class* or the *concept class* $\mathcal{H}$ over the instance space $\mathcal{X}$. The hypothesis class over $\mathcal{X}$ is defined as the space of functions from $\mathcal{X}$ to $\mathcal{Y}$ i.e. $f : \mathcal{X} \to \mathcal{Y}$. An element of this set is referred to as a hypothesis or a concept. Examples of $\mathcal{H}$ include all linear separators and all neural networks that have a fixed architecture. For example, the class of multi-layer perceptrons defined below is an example of a hypothesis class and one that we will use throughout the rest of the thesis.

Definition 1 (Multi-Layer Perceptron)**.** *A Multi-Layer Perceptron (MLP) is a hierarchial model with a sequence of L layers. The i^{th} layer is parameterized by a matrix $\mathbf{W}_i \in \mathbb{R}^{\ell_{i-1} \times \ell_i}$ where ℓ_i represents the width of the i^{th} layer and $W = \max_{1 \leq L} \ell_i$ is the width of the MLP. The hypothesis class of MLPs $\mathcal{M}$ is*

$$\mathcal{M} : \{h_\theta \mid h_\theta(\mathbf{x}) = \mathbf{W}_1 \phi_1 (\cdots \phi_{l-1} (\mathbf{W}_{L-1} \phi_L (\mathbf{W}_L \mathbf{x}))), \theta = \{\mathbf{W}_1, \cdots, \mathbf{W}_L\}\}$$

Each ϕ_i is an activation function and can all be identical or different depending on the network configuration. In practice, they are usually the same. When ϕ operates on a vector, it operates element-wise on each element of the vector.

Finally, we also define the *loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, which is used to measure the goodness of the functions in the hypothesis class. Intuitively, a loss function denotes the price we pay when a function $f \in \mathcal{H}$ sees $\mathbf{x} \in \mathcal{X}$ and guesses the output to be $f(\mathbf{x}) \in \mathcal{Y}$ when it is actually $y \in \mathcal{Y}$. The price paid is small when $f(\mathbf{x})$ and y are close and large when they are far. The goodness of the function $f \in \mathcal{H}$ on the whole distribution is measured with the notion of Expected Risk.

Definition 2 (Expected risk)**.** *For a distribution $\mathcal{D}$, loss function ℓ, and a hypothesis $h \in \mathcal{H}$, the expected risk is defined as*

$$\mathcal{R}(h; \mathcal{D}, \ell) = \int_{\mathbf{z}=(\mathbf{x},y)\sim\mathcal{D}} \ell(h(\mathbf{x}), y)\ dD(\mathbf{z})$$

When we omit ℓ from the definition of the expected risk, we will refer to the expected error for the $0/1$ loss function $\ell_{0,1}(y, \hat{y}) = \mathbb{I}\{y \neq \hat{y}\}$

$$\mathcal{R}(f; \mathcal{D}) = \mathcal{R}(h; \mathcal{D}, \ell_{0,1}) = \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}[f(\mathbf{x}) \neq y],$$

Now, we are ready to define the learning problem. Given a hypothesis class $\mathcal{H}$, a loss function ℓ, and a training set $\mathcal{S}_m$ consisting of m i.i.d points sampled from an unknown but fixed *data distribution* $\mathcal{D}$, the objective of a learning algorithm $\mathcal{A}$ is to return a hypothesis $h = \mathcal{A}(\mathcal{S}_m) \in \mathcal{H}$ that, with high probability, minimizes the expected risk $\mathcal{R}(h; \mathcal{D}, \ell)$. For simplicity, we will omit $\mathcal{D}$ and ℓ from the definition of expected risk when they will be clear from the context.

2.1.1 PAC Learning

We will provide the formal definition of the learning problem using the Probably Approximate Correct (PAC) [262] learning framework. Let the distribution $\mathcal{D}_\mathcal{X}$ be the restriction of the distribution $\mathcal{D}$ on the instance space $\mathcal{X}$ such that there is a *target* concept class $\mathcal{C}$ along with a target concept $c \in \mathcal{C}$, which labels the data

drawn from $\mathcal{D}_{\mathcal{X}}$. The objective of the learning problem is to capture c as closely as possible. To highlight that the generation of the training dataset is a random process and that depends on the target concept c, the training dataset is obtained as follows. First, an instance $\mathbf{x} \in \mathcal{X}$ is sampled from $\mathcal{D}_{\mathcal{X}}$ and is then assigned a label by the target concept c. This process is referred to as a call to the example oracle, denoted as $EX\left(c; D_{\mathcal{X}}\right)$. Therefore, drawing an m-sized dataset $\mathcal{S}_m$ can be simulated with m calls to $EX\left(c; \mathcal{D}_{\mathcal{X}}\right)$.

Definition 3 (PAC-Learnability)**.** *A concept class $\mathcal{C}$ is said to be PAC-learnable using the hypothesis class $\mathcal{H}$, if there exists a (possibly randomized) algorithm $\mathcal{A}$ such that the following holds true. For every $c \in \mathcal{C}$, for every distribution $\mathcal{D}_{\mathcal{X}}$ over $\mathcal{X}$, for every $0 < \epsilon, \delta < 1$, if $\mathcal{A}$ is given access to $EX\left(c; \mathcal{D}_{\mathcal{X}}\right)$ and is given as input ϵ, δ, then $\mathcal{A}$ makes m calls to $EX\left(c; \mathcal{D}_{\mathcal{X}}\right)$ and returns $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, $\mathcal{R}(h; \mathcal{D}) \leq \epsilon$. The probability is over the randomization in the calls to the example oracle and the internal randomization of $\mathcal{A}$.*

The number of calls made to $EX\left(c; \mathcal{D}_{\mathcal{X}}\right)$ is referred to as the sample complexity (denoted by m) and must be bounded by a polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}$, and some parameters depending on the size of c and the size of the instance space $\mathcal{X}$. Further, all hypotheses in $\mathcal{H}$ must also be evaluable in polynomial time (in the size of the input).

Without diving too deep into the details of PAC Learning[1], we will discuss the importance of the various components of the PAC learning framework. It is important to note at this stage that our main goal in this thesis is to control not just the classification error, but also to improve certain additional properties of reliability (which will be discussed in detail in Chapter 3) in machine learning. Thus, our discussion will diverge, at places, from the original PAC learning framework even though the essential components like sample complexity, error and confidence parameter and choice of hypothesis class remain equally relevant. Below, we discuss some of the important points.

1. The sample complexity for learning should be relatively small i.e. polynomial in problem parameters. When learning from real-world data, it is usually

[1]For a more detailed expose into this topic, please refer to the excellent lecture notes in `http://www.cs.ox.ac.uk/people/varun.kanade/teaching/CLT-HT-TT2021/lectures/CLT.pdf`

not possible to see more than a fixed number of samples, so it might not be practical to expect our learning algorithm to achieve arbitrarily low error or very high confidence in the low data regime. Thus, it is important to understand what error and confidence are achievable in the low data regime.

Further, when along with classification error the learning problem also requires controlling an additional metric of reliability like adversarial vulnerability or mis-calibration, the sample complexity of the problem might increase. We will identify theoretically and experimentally when this increase in sample complexity happens and if necessary, design learning algorithms where this increase is small or absent.

2. Choosing the hypothesis class $\mathcal{H}$ that is used to learn the target concept c is perhaps the most relevant component of PAC learning for this thesis. This is important for multiple reasons. First, the sample complexity of learning the same target concept can be large or small depending on the choice of hypothesis class it is being learnt from. This is discussed further in Section 2.2.

   Second, we are looking to control not just the classification error but also other metrics of reliability. Depending on the choice of hypothesis class, it might be impossible to control both the test error and the metric of reliability simultaneously. Even in situations where they can both be controlled simultaneously, it might lead to an increase in sample complexity. We will investigate which combined choice of an additional metric of reliability and hypothesis class can pose such problems.

2.1.2 Learning with noise

One of the important restrictions of the PAC learning framework, as defined in Definition 3, is the assumption that the example oracle always faithfully returns unblemished examples drawn from the target distribution and labels them according to the target concept. Often, for the data given to training algorithms, this is too strict a requirement as label noise is ubiquitous in real-world data. Such noise can arise from multiple sources including a malicious or careless data annotator, faulty communi-

cation equipment transmitting the data, or faulty recording equipment recording the data. This necessitates the design of a formal framework that can capture noise in the data generation process. The noisy PAC learning framework of Angluin and Laird [12] is an example of such a framework.

The important change from the noiseless framework is in the way data is generated using the example oracle. Let $0 \leq \eta < \frac{1}{2}$ be the noise rate. A simple framework to simulate the noise is through the Random Classification Noise (RCN) oracle denoted as $EX_\eta(c; \mathcal{D}_\mathcal{X})$. To generate an instance using this oracle, first, an instance is generated using the original example oracle $EX(c; \mathcal{D}_\mathcal{X})$ and then with probability η the label is flipped to an arbitrary incorrect label.

PAC-Learnability with Random Classification Noise is defined exactly as PAC Learnability but with the example oracle $EX(c; \mathcal{D}_\mathcal{X})$ replaced with the noisy example oracle $EX_\eta(c; \mathcal{D}_\mathcal{X})$ and $\frac{1}{1-2\eta}$ added to the set of parameters the sample complexity should be polynomial in.

Definition 4 (PAC-Learnability with Random Classification Noise)**.** *A concept class $\mathcal{C}$ is said to be PAC-learnable in the presence of random classification noise using the hypothesis class $\mathcal{H}$, if there exists a (possibly randomized) algorithm $\mathcal{A}$ such that the following holds. For every $c \in \mathcal{C}$, for every distribution $\mathcal{D}_\mathcal{X}$ over $\mathcal{X}$, for every $0 < \epsilon, \delta < 1$, and for every $0 \leq \eta < \frac{1}{2}$, if $\mathcal{A}$ is given access to $EX_\eta(c; \mathcal{D}_\mathcal{X})$ and is given as input ϵ, δ and η, then $\mathcal{A}$ makes m calls to $EX_\eta(c; \mathcal{D}_\mathcal{X})$ and returns $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, $\mathcal{R}(h; \mathcal{D}) \leq \epsilon$, where the probability is over the randomization in the calls to the example oracle and the internal randomization of $\mathcal{A}$.*

The number of calls made to $EX(c; \mathcal{D}_\mathcal{X})$, referred to as the sample complexity (denoted with m), must be bounded by a polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-2\eta}$ and some parameters depending on the size of c and size of the instance space $\mathcal{X}$. Further, all hypotheses in $\mathcal{H}$ must also be evaluable in polynomial time (in the size of the input).

While the presence of Random Classification Noise (RCN) is an added challenge, several hypothesis classes that are PAC learnable in the original definition of PAC learning (c.f. Definition 3) have also been shown to be PAC learnable with random

classification noise. One of the only problems for which no algorithm is known in the noisy setting of Definition 4 but is known to be learnable in the noiseless PAC setting of Definition 3 is the parity problem [138]. However, even for problems that are learnable in the noisy setting, the learning algorithm might be different from the noiseless setting. In Chapter 4, we discuss and provide experimental evidence for how a learning algorithm can impose additional structure on the hypothesis class to aid learning in the presence of random classification noise.

When the learning task must also satisfy additional metrics of reliability like robustness and calibration, the presence of random classification noise can pose further challenges. In Chapter 5, we discuss this for the specific case of adversarial robustness.

2.2 Empirical Risk Minimisation

The most common approach to solving the learning problem described above is the Empirical Risk Minimisation [264] approach. In this approach, instead of minimizing the expected risk, which is harder as $\mathcal{D}$ is unknown, the learning algorithm instead minimises the empirical risk $\widehat{\mathrm{R}}_N(h, \ell)$ (defined in Definition 5) on the observed training dataset. Like the expected risk, we will ignore ℓ from the definition of empirical risk when it will be clear from context[2].

Definition 5 (Empirical risk)**.** *If the training set consists of* $\{(\mathbf{x}_i, y_i)\}$ *for* $i \in \{1 \cdots N\}$ *then the empirical loss is defined as*

$$\widehat{\mathrm{R}}_N(h) = \frac{1}{N} \sum_{i=1}^{N} \ell(h(\mathbf{x}_i), y_i)$$

The theory of minimizing this empirical risk is usually tackled via the theory of optimisation. The main purpose of the theory of optimisation is to find a model h and guarantee that for some small ϵ, the empirical risk of h is close to the best

[2]Please refer to [265] for a more detailed description of statistical learning theory that discusses the properties of minimizing the empirical risk.

possible empirical risk i.e.

$$\widehat{\mathrm{R}}_N(h) - \min_{h \in \mathcal{H}} \widehat{\mathrm{R}}_N(h) \leq \epsilon.$$

However, the task of *learning* requires us to perform well not only on the seen dataset $\mathcal{S}_N$ but also on unseen data from the entire distribution $\mathcal{D}$, which is measured using the expected risk from Definition 2. The theory of generalisation guarantees with a high probability, that for any hypothesis h in the hypothesis class $\mathcal{H}$, the empirical and the expected risk are close i.e.

$$|\widehat{\mathrm{R}}_N(h) - \mathcal{R}(h)| \leq \sup_{h \in \mathcal{H}} |\widehat{\mathrm{R}}_N(h) - \mathcal{R}(h)| = \zeta(N; \mathcal{H}),$$

where $\zeta(N; \mathcal{H})$ decreases with increasing N.

The main focus of the theory is two-fold:

- To guarantee that with an increasing number N of observations, the difference of the two quantities goes to zero for any model i.e. $\lim_{N \to \infty} \zeta(N, \mathcal{H}) \to 0$.
- To estimate the difference $\zeta(N; \mathcal{H})$, when only a finite number N of samples are available.

It turns out that the second property is more important for our purposes when dealing with neural networks. Not only does it have more practical relevance, but while the first property almost always holds [3], the second property is mathematically more complicated especially for complex hypothesis classes like neural networks. For example, if the empirical and expected risks are bounded between 0 and 1, which is a very reasonable assumption, then any value of $\zeta(N; \mathcal{H})$ that is greater than 1 is a vacuous upper bound. The main goal of generalisation theories is to get accurate estimates of $\zeta(N, \mathcal{H})$ for moderate values of N and some useful hypothesis class $\mathcal{H}$. This is usually done through theorems like Theorem A referred to as the *basic theorem of generalisation*. However, this is not necessarily the only way (c.f. Bousquet and Elisseeff [34]).

[3] The asymptomatic behaviour of the empirical error functions is captured by the Glivenko–Cantelli theorem as the number of i.i.d samples goes to infinity

Theorem A (Basic Theorem of Generalisation)**.** *Let $\mathcal{H}$ be a set of hypotheses from the space $\mathcal{X}$ to $\{0,1\}$ and the rest of the terms be as defined in the previous sections. Then $\forall \delta \in (0,1)$, with a probability of at least $1-\delta$, $\forall h \in \mathcal{H}$*

$$\left|\widehat{\mathrm{R}}_N(h) - \mathcal{R}(h)\right| \leq \zeta(N;\mathcal{H}) = \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right),$$

where $C(\mathcal{H})$ denotes a measure of complexity for the hypothesis class $\mathcal{H}$[4]. $\tilde{\mathcal{O}}(\cdot)$ is a big-Oh notation that omits logarithmic terms.

Using this theorem, we can argue that empirical risk minimisation yields a hypothesis whose expected risk is close to the best possible expected risk as long as the complexity of the classifier is low. Let the hypothesis returned by the algorithm $\mathcal{A}$ be $h_L \in \mathcal{H}$. We know that $\widehat{\mathrm{R}}_N(h_L) \leq \min_{h\in\mathcal{H}} \widehat{\mathrm{R}}_N(h) + \epsilon$ for some ϵ. Define $h^* = \mathbf{argmin}_{h\in\mathcal{H}} \mathcal{R}(h)$.

According to Theorem A, with probability at least $1-\frac{\delta}{2}$ each of Equations (2.1) and (2.2) holds

$$\mathcal{R}(h_L) - \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right) \leq \widehat{\mathrm{R}}_N(h_L) \leq \mathcal{R}(h_L) + \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right) \tag{2.1}$$

and

$$\mathcal{R}(h^*) - \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right) \leq \widehat{\mathrm{R}}_N(h^*) \leq \mathcal{R}(h^*) + \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right). \tag{2.2}$$

Thus,

$$\begin{aligned}
\mathcal{R}(h_L) &\leq \widehat{\mathrm{R}}_N(h_L) + \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right) \\
&\leq \min_{h\in\mathcal{H}} \widehat{\mathrm{R}}_N(h_L) + \epsilon + \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right) \\
&\leq \widehat{\mathrm{R}}_N(h^*) + \epsilon + \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right) \\
&\leq \mathcal{R}(h^*) + \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right) + \epsilon + \tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right).
\end{aligned}$$

[4]For example, if $\mathcal{H}$ is a finite set of hypotheses, then the cardinality function is a valid instance of $C(\cdot)$. We will look into more examples of the complexity function in Chapter 3.

Therefore, with probability at least $(1 - \delta)$

$$\mathcal{R}(h_L) - \mathcal{R}(h^*) \leq 2\tilde{\mathcal{O}}\left(\frac{\sqrt{C(\mathcal{H})}}{N}\right) + \epsilon. \tag{2.3}$$

As discussed before, it is the purpose of optimisation algorithms to minimise ϵ. In fact, neural networks usually obtain $\epsilon = 0$ as has been shown empirically in multiple papers including the seminal paper of Zhang et al. [292]. So, the real difficulty is in coming up with estimations of $C(\mathcal{H})$, that are practically useful and not vacuous.

2.2.1 ERM in the presence of Label Noise

Label noise is ubiquitous in real-world data. In fact, common datasets like MNIST, CIFAR10, and CIFAR100 contain label noise in the training dataset as we show in Chapter 5. In Definition 4, we discussed a variant of PAC learning which allows for noise in the data generation process and we made a note that most problems that are learnable in the noiseless setting are also learnable in the noisy setting. However, even in cases where the problem is learnable in the noisy setting, it does not automatically imply that the algorithm for learning in the noiseless setting is still adequate for learning in the noisy setting. Therefore, understanding whether algorithms and frameworks that are useful for learning in the noiseless setting can also be used in the noisy setting and identifying the associated risks is important for deploying learning algorithms in the real world.

Empirical Risk Minimisation (ERM) is the most commonly used framework for learning in the noiseless setting and enjoys formal generalisation guarantees as shown in Equation (2.3). However, the guarantees do not immediately apply to learning in the presence of noise. To see this, note that the distribution, the noisy example oracle samples from in Definition 4, is different from the original distribution $\mathcal{D}$. Belkin et al. [26] provide more rigorous arguments for why under conventional statistical generalisation theory, one would expect empirical risk minimisation on noisy data to cause generalisation error to increase rapidly with increasing noise. They show that for a certain class of kernel classifiers, the RKHS norm of any classifier

that overfits noisy training data grows nearly exponentially with dataset size[5]. As most generalisation bounds for kernel classifiers depend at most polynomially on the RKHS norm of the classifier, they diverge to infinity as the dataset size increases.

However, Belkin et al. [26] observe empirically that the generalisation performance of these classifiers does not deteriorate as quickly in practice as the existing theory suggests. They observe that as training loss tends towards zero even in the presence of label noise, test error either remains stable or decreases and then stabilises. Similar behaviour, albeit in the less rigorous setting of over-parameterised deep neural networks, has been observed in Zhang et al. [292] where deep neural networks were trained on CIFAR10 with varying levels of random classification noise. The results reported in Figure 1(c) in Zhang et al. [292] show that when neural networks fit the noisy dataset, even for large noise levels, test accuracy remains high on clean test-sets.

Some recent works look at characterizing learning problems where fitting noisy labels does not cause the test error to blow up, a phenomenon referred to as *benign overfitting*. Chatterji and Long [47] show that if a linear classifier fits noisy data then the test error will be close to the noise rate (i.e. have a decent test accuracy) only if the model is highly over-parameterised. In Theorem 3.1, they show that, under some conditions, the error rate is upper bounded by (label noise rate (η)) + (a term depending on $e^{-p/C}$), where p is the dimensionality of the parameters and C is a constant. Thus for small p, the model that overfits the noisy training set can have a very large test error whereas for large p, this error will be close to the noise level. Bartlett et al. [23] look at this in the setting of linear regression. In Theorem 4, they provide matching lower and upper bounds for the test risk when the linear regression achieves zero regression error. Their result shows that the dimensionality of the problem determines whether the test risk will be good for a regressor that perfectly fits a noisy training set. Thus, depending on the properties of the problem such as its dimensionality, the perfectly fit model may get very bad or very good test loss.

These works provide support for the applicability of ERM in the presence of

[5]Note that for a constant noise rate the amount of noise increases with increasing dataset size

noisy labels for large deep neural networks and highly over-parameterised models in general. However, even in the favourable scenario where ERM with noisy labels yields a generalisable model, the model may suffer in other notions of reliability that we care about. In Chapters 5 and 7, we show how empirical risk minimisation can provide good test error and yet suffer in adversarial robustness and calibration.

2.3 Regularisation

To guarantee generalisation of models based on empirical error using the Basic Theorem of Generalisation (Theorem A), one solution is obviously to just obtain a large amount of data (i.e. large N). However, often this is infeasible as data collection is expensive, time-consuming, and sometimes it is impossible.

On the other hand, if we have the right definitions of the complexity measure C in Theorem A, we can prescribe the amount of data required to guarantee generalisation. Similarly, if a limited amount of data is available, then one can prescribe the maximum value $r \in \mathbb{R}_+$ of the complexity measure C for which the available data is sufficient to guarantee generalisation. In that case, the search of hypothesis can be restricted to a subset $H' \subseteq \mathcal{H}$ that satisfies the prescribed constraint on model complexity i.e. $C(\mathcal{H}') \leq r$. It is this problem of restricting the search of the hypothesis that regularisation tries to solve.

Regularisation can be implemented through a regularisation function $\Omega : \mathcal{H} \to \mathbb{R}_+$ that assigns a complexity value to every hypothesis h in $\mathcal{H}$. One can think of Ω as a proxy of the complexity function C in Theorem A. Note that schemes like early stopping in gradient-based training, that do not directly use a regularisation function, are also referred to as regularisation because they have been shown to control some kind of complexity like a regularisation function [284]. In the rest of this section, we will see different notions of optimisation with regularisations i.e. different ways of imposing the constraints on the identified structures in neural networks.

Ivanov regularisation

The purpose of regularisation is to restrict the hypothesis space $\mathcal{H}$ by imposing further constraints. For example, if $\mathcal{H}$ were originally the class of linear predictors in $\mathbb{R}^2$, a valid form of regularisation would be to consider only those linear predictors that are parallel to one of the coordinate axes. This particular example is called ℓ_0 regularisation. Effectively, this reduces the complexity of $\mathcal{H}$. In machine learning literature, such a regularisation is referred to as Ivanov regularisation. Definition 6 defines an Ivanov-regularised ERM problem.

Definition 6 (Ivanov regularisation)**.** *Let $\Omega : \mathcal{H} \to \mathbb{R}_+$ be the regularisation function and $r \geq 0$ be the maximum allowed value of the regularisation function. Then the regularised ERM problem is defined as obtaining a hypothesis h_r^*:*

$$h_r^* = \min_{h \in \mathcal{H}} \widehat{\mathrm{R}}_N(h)$$
$$s.t.\ \Omega(h) \leq r$$

In Neural networks, regularisations like spectral normalization [182] and Parseval networks [55] can be thought of as an attempt to solve this problem[6]. Some prominent instances are problems in low rank matrix approximation [134], matrix recovery [42],and sparse recovery [32]. In Chapter 4, we devise a novel Ivanov regularisation called Stable Rank Normalization (SRN). Interestingly, we also obtain a closed-form optimal solution to the projection problem which is not very common among problems of this type which are solved only approximately.

In terms of generalisation theory, solving an Ivanov-regularised optimisation problem can immediately provide meaningful guarantees on generalisation error. If the capacity function C is monotonically dependent on the regularisation function Ω, then constraining Ω provides an upper bound on the capacity C which can be used to compute the generalisation error. The problem referred to above and in Chapter 4 is motivated by this idea.

[6] I have referred to this as an attempt because they solve the problem of projection onto the set of classifiers with correct spectral norm approximately

Tikhonov regularisation

The more commonly used regularisation in machine learning and particularly gradient-based learning is Tikhonov regularisation defined below in Definition 7.

Definition 7 (Tikhonov regularisation)**.** *Let $\Omega : \mathcal{H} \to \mathbb{R}_+$ be the regularisation function and $\lambda \geq 0$ be the regularisation coefficient. Then the Tikhonov regularised ERM problem is defined as obtaining a hypothesis h^*_λ as follows*

$$h^*_\lambda = \min_{h \in \mathcal{H}} \widehat{\mathrm{R}}_N(h) + \lambda \Omega(h)$$

Regularisers like weight decay [146], L_1 regularisation [77, 247], and spectral regularisation [287] are examples of Tikhonov regularisation in deep learning. While Ivanov and Tikhonov regularisations are equivalent for some value of the coefficients, the main advantage of Tikhonov regularisation is that it converts a constrained optimisation problem to an unconstrained optimisation problem. Therefore, it is usually enough to apply gradient-based optimisation methods. However, it must be noted that this is not ERM though it is sometimes referred to as regularised Empirical Risk Minimisation.

Morozov regularisation

Another form of commonly studied regularisation is residual learning or also referred to as Morozov regularisation [187]. The main objective here is to minimise the regularisation function with constraints on the original ERM loss. This is common in settings where the noise rate of the problem is known and hence expecting a loss value lesser than the noise rate would be unreasonable.

Definition 8 (Morozov regularisation)**.** *Let $\Omega : \mathcal{H} \to \mathbb{R}_+$ be the regularisation function and $\delta \geq 0$ be the discrepancy parameter. Then the Morozov regularised ERM problem is defined as obtaining a hypothesis h^*_ϵ that satisfies*

$$h^*_\delta = \min_{h \in \mathcal{H}} \Omega(h)$$
$$s.t.\ \widehat{\mathrm{R}}_N(h) \leq \delta$$

2.3.1 Data-dependent regularisation

Common examples of regularisations usually comprise data-independent regularisations. In the formulation of Definitions 6 to 8, the value of the regulariser term $\Omega(h)$ does not depend on the data directly unlike the empirical loss term $\widehat{\mathrm{R}}_N(h)$. Thus, the penalisation (Definitions 7 and 8) or the feasibility (Definition 6) of a particular hypothesis h is the same regardless of the properties of the dataset. In some sense, these regularisers solve learning problems with little or no assumptions about the data distribution.

However, there are cases when information about the data-generating distribution needs to be accounted for. As a toy example, consider a high dimensional instance space where the information in the data is captured in a small number of principal input features and the remaining nuisance features contain no information. The purpose of the regulariser, in this problem, is to find a low complexity model that fits the observed dataset from this distribution. As it is known that the unseen test data from this distribution will also not have any weight in the nuisance features, two models that behave identically on the principal components but differently on the nuisance components should not be penalised differently by the regulariser. However, to do this the regulariser needs to be aware of the data distribution to identify the principal and nuisance components.

In practice, one common setting where data-dependent regularisers are applied is semi-supervised learning problems where most data samples have no labels and thus further information about the data distribution is required to relate available information from the labelled data to predict labels for the unlabelled data. Information regularisation [57] enforces this by assuming that label predictions can be made by clustering points and that each cluster is somewhat pure in its label. Their algorithm can be cast as a modified data-dependent Tikhonov regularisation. Another prominent example of data-dependent Tikhonov regularisation includes maximizing the entropy of the predicted class-label distribution [209] to reduce the confidence of the model on incorrect predictions.

In Chapter 6, we look at an optimisation problem with data-dependent Ivanov

regularisation. In particular, we look at constraining the rank of the representation space of deep neural networks. The intuition is that the model should exploit as small a number of features as possible to do the classification, as unnecessary features can be exploited by an adversary to construct adversarial attacks. We solve the problem by converting it to an equivalent data-dependent Tikhonov regularisation problem. In Chapter 7, we discuss Focal Loss which is a modified loss function but that can be shown to be also acting as maximizing the entropy of the predicted class probabilities, inherently doing a form of data-dependent Tikhonov regularisation. Further, while the stable rank normalization presented in Chapter 4 is an Ivanov regularisation approach, we also discuss its impact on some data-dependent Lipschitz terms.

3. Background for Reliable Deep Learning

Algorithms to learn from data are now widely applied to make our lives easier. Applications of these systems include medical diagnoses [144, 31], fraud detection from personal finance data [87], and online community detection from user data [81]. They also governs our online presence by recommending what content we should see on social networks, who we should connect with on professional networks, and which words we should use to complete our emails. At the core, these are all instances of algorithms that have been learnt from past data to maximise some metric chosen by the service providers that deploy them. These metrics include accuracy of classification (eg. in medical diagnosis and fraud detection), engagement (eg. in social media and professional media recommendations) and/or revenue.

However, as these algorithms are deployed at large scale it is becoming increasingly clear that they suffer from some pathological issues. In particular, deep learning, with its huge success both in terms of performance and the fast pace at which it is being adopted by industry, poses a particularly critical challenge as we are still unaware of the extent of its vulnerabilities. In some cases, as the later chapters demonstrate, the pursuit of higher performance under the original metrics further aggravates these vulnerabilities. If such pursuit remains unchecked, it will lead to a decrease in trust in this technology and, possibly, even have an adverse effect on the proper functioning of society. This chapter gives an overview of four such issues that are particularly critical for deep neural networks. In addition, it discusses past works in these fields and how the rest of the thesis fits into the context of these works.

3.1 Generalisation in Neural Networks

A machine learning model that achieves a very small error on the observed training set but fails to perform when tested on unseen data is of little use. In fact, such a model might be dangerous if the user is unable to distinguish models that perform

well on both the observed training set and unseen test data from models that perform well on training sets but fail on test data. The first kind of model, discussed extensively in Section 2.1, is said to have generalised whereas the second model is said to have failed at generalisation.

For neural networks, this is particularly worrisome. Consider the following thought experiment with a data distribution over natural images equally distributed in k classes and two target concepts (labelling functions): one of which labels each image correctly and the other which labels each image randomly, The two training datasets are constructed by first sampling N points independently from the distribution and then labelling the first dataset with the *correct* target concept and the second dataset with the *random* target concept. A sufficiently large and properly trained neural network is able to achieve zero training error via ERM for both of these training datasets as seen from the experiments in Zhang et al. [292] and further in Chapter 4. Further, the first network would also generalise to unseen data from that distribution labelled by the *correct* target concept. On the other hand, for the second concept, the target labelling concept is a random labelling function. Hence, no model can achieve an expected risk that is better than what can be achieved with random guesses. Thus, two neural networks with the same architecture achieve the same training error on two different learning tasks but while one of the them generalises, the other does not. This shows that given a neural network architecture and a training dataset, it is not possible to certify, whether the neural network trained on that dataset will generalise without any further information about the data distribution. This tendency of neural networks to obtain zero training error via ERM, irrespective of their test error, makes it ever more important to have a rigorous generalisation theory for them. The gold standard would be a theory which, given a model architecture, learning algorithm, and a dataset, is able to certify with a very high confidence whether the model generalises.

3.1.1 Importance of structures for generalisation in neural networks

Guaranteeing generalisation for models obtained via ERM using results akin to Equation (2.3) requires

1. an upper-bound on the true complexity of the model and
2. a dataset, whose size is proportionally large to match the upper-bound on model complexity.

If the proposed upper-bound on the model complexity is loose, the corresponding number of samples required to theoretically guarantee generalisation using Theorem A will be large even if the true complexity of the model is low and thus, in practice, generalisation will be observed with a smaller dataset.

In the case of neural networks, it has been difficult to find accurate definitions of the complexity measure C in Theorem A. In fact, definitions of C from most recent works prescribe a value of N that is too large to justify the generalisation we see in real life with an amount of data that is orders of magnitude less than the prescribed amount. Arora et al. [16] shows (See Figure 3 in Arora et al. [16]) that when recent measures of complexity [21, 195, 22, 196, 16] are applied to VGG-19, the resultant prescribed sample complexities are many orders of magnitude larger than the actual number of trainable parameters, let alone the number of training examples. Dziugaite and Roy [74] also show (see Appendix D in Dziugaite and Roy [74]) that Rademacher complexity bounds computed using the complexity bound of Neyshabur et al. [195] are vacuous for deep neural networks i.e. the bounds predict that the test error will be less than one, which is trivially satisfied by the definition of test error.

While existing works in generalisation theory for neural networks seem to be unable to prescribe the absolute amount of data (even approximately) required for generalisation in practice, it is natural to ask whether the *structures*, identified by these works as being important for generalisation, do in fact causally impact gen-

eralisation in practice[1]. To address this, Jiang* et al. [132] conduct a large-scale empirical study to find out if there are causal relationships between recently proposed complexity measures and generalisation in neural networks. They train more than 2000 models on CIFAR10 in a controlled setup by systematically varying important hyper-parameters, optimisation algorithms, and stopping criterion, and investigate whether there is a strong correlation between generalisation and any of the over 40 complexity measures that they study. Their results indicate that a large number of them do indeed appear to be causally related to generalisation even though the bounds themselves are vacuous i.e. the test error of different learnt models maintain the same ordering as predicted by applying the generalisation bounds on the the learnt models though the exact upper-bound on the test error predicted by the generalisation bounds are greater than one, which is true by the definition of test error. Despite the bounds being vacuous, the causal relationship between the generalisation error and the complexity bounds explored in their work suggests that regularising these complexity measures might benefit generalisation in practice.

However, their study also suggests, somewhat counterintuitively and without a causal explanation, that a large number of these complexity measures are negatively correlated with generalisation. One of the limitations of their study in identifying correlation between complexity measures and generalisation is that they do not explicitly penalise the complexity measures in their experiments. Without explicit penalties, the complexity measures of the learnt models have higher magnitudes than what would be the case with said penalty. They compute the correlation between generalisation and complexity measures by studying only the distribution on the complexity measures induced by training without explicit penalties (natural training). Thus, it is possible that when the complexity measures lie in a range of larger magnitudes via natural training, they do not show a strong correlation with generalisation but lower magnitudes obtained through direct penalisation will show a strong correlation. We show in Chapter 4 that explicit penalisation of the structures identified by these complexity measures indeed shows a greater impact

[1]In this thesis, we use the term *structure* to loosely refer to properties of both — neural networks and the data distribution eg. rank of the weight matrices, entropy of the probability distribution over classes, rank of representations, and noise in data.

on the generalisation behaviour in practice.

Thus our focus in this thesis is on studying the structures identified by these complexity bounds and designing optimisation algorithms or regularisation techniques to constrain them in practice in an efficient way. As the main purpose of this thesis is not to propose new tighter generalisation bounds, we will not delve deep into defining the complexity measure C. However, a general introduction to this topic is necessary to demonstrate the context and significance of some of the future chapters, especially Chapters 4 and 6.

3.1.2 Norm-based complexity measures

Ultimately, the objective of designing complexity measures for neural networks is to use them in some variant of Theorem A and obtain generalisation guarantees for these networks. One well-known instantiation of the theorem involves Rademacher complexities (Definition 9). Rademacher complexity [143] is a relatively modern notion of complexity that is (data) distribution dependent and is defined for any class of real-valued functions. The Rademacher complexity of a hypothesis class $\mathcal{H}$ over sets, of size N, drawn i.i.d from a distribution $\mathcal{D}$ ($\mathcal{D}$ is usually omitted from the notation) is denoted by $\mathrm{RAD}_m(\mathcal{H})$.

Definition 9 (Rademacher Complexity [143])**.** *Let a sample $\mathcal{S} = \{\mathbf{x}_1, \cdots, \mathbf{x}_m\}$ be drawn i.i.d. from a data distribution $\mathcal{D}_{\mathcal{X}}$ over the instance space $\mathcal{X}$ and consider a hypothesis class $\mathcal{H}$, then the* empirical rademacher complexity *of $\mathcal{H}$ on the sample $\mathcal{S}$ is defined as*

$$\widehat{\mathrm{RAD}}_m(\mathcal{H}) = \mathbb{E}\left[\sup_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i h(\mathbf{x_i})\right] \tag{3.1}$$

where the expectation is over the rademacher variables σ_i, which are independent, uniform, $\{\pm 1\}$ valued random variables

The rademacher complexity is then defined as the expectation of the empirical rademacher complexity over the selection of the sample set using the underlying distribution.

$$\mathrm{RAD}_m(\mathcal{H}) = \mathbb{E}_{\mathcal{S}\sim\mathcal{D}_{\mathcal{X}}^m}\left[\widehat{\mathrm{RAD}}_m(\mathcal{H})\right] \tag{3.2}$$

Recall the second learning task in the thought experiment from Section 3.1 where

the target concept is a random labelling function. Clearly, this is a meaningless learning problem. The empirical rademacher complexity in Definition 9 captures (in expectation over the labelling) how well the best hypothesis from the hypothesis class $\mathcal{H}$ can fit these random labels on the given dataset. In other words, it measures how well $\mathcal{H}$ can fit noise. Experiments in Zhang et al. [292], showing that certain classes of deep neural networks like ResNet can memorise noise, suggests that these classes have a very large empirical rademacher complexity.

Theorem B (Rademacher Complexity Generalisation Bound)**.** *Consider a hypothesis class $\mathcal{H}$ of hypothesis $h : \mathbb{R}^d \to [0, 1]$ and, $\mathcal{R}(h)$ and $\widehat{\mathrm{R}}_N(h)$ are the expected and empirical risks as defined in Definitions 2 and 5. Then $\forall \delta \geq 0$, with probability $1 - \delta$,the following holds*[2] *$\forall h \in \mathcal{H}$*

$$\mathcal{R}(h) \leq \widehat{\mathrm{R}}_N(h) + 2\mathrm{RAD}_N(\mathcal{H}) + \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2N}}$$

To use Theorem B for neural networks, we need to compute the rademacher complexities of classes of neural networks. A series of papers [21, 195, 91] has designed Rademacher complexity bounds for neural networks. These bounds primarily depend on the product of the Frobenius norm of the weight parameters of each layer $\|\mathbf{W}_\mathrm{i}\|_\mathrm{F}$ and the depth of the network L. As Theorem C shows, some of these bounds (c.f. Equation (3.3)) have an exponential dependence on network depth L while others [91] avoid the exponential dependence but still have an indirect dependence on the network depth through the product of the norms. Note that the only term in Theorem C that contains learnable parameters and can be controlled through regularisation are the Frobenius norms $\|\mathbf{W}_\mathrm{i}\|_\mathrm{F}$ of each weight matrix. Thus, explicit regularisation of the norms of each weight parameter in a neural network through regularisers like weight decay might help with generalisation. Experiments in Zhang et al. [292] (c.f. Table 1,2 in Zhang et al. [292]) show that while this has a positive effect on generalisation, the effect is small. Thus, there is a need to identify further structures which have stronger impact on generalisation.

[2]Proof can be found in `http://www.cs.ox.ac.uk/people/varun.kanade/teaching/CLT-HT2018/lectures/lecture08.pdf`.

Theorem C (Simplified Rademacher Complexity of MLP [21, 195, 91])**.** *Consider a class of real-valued Multi-Layer Perceptrons (Definition 1) $\mathcal{N}$ with ReLU activation function and L layers. The Rademacher complexity of $\mathcal{N}$ scales as follows [195]*[3]

$$\widehat{\mathrm{RAD}}_N(\mathcal{N}) \leq \frac{1}{\sqrt{N}} 2^{L+1} \left(\prod_{j=1}^{L} \|\mathbf{W}_j\|_F \right) \left(\max_{x \in \mathcal{X}} \|x\| \right) \tag{3.3}$$

where $\mathbf{W}_j$ is the weight parameter for the j^{th} layer of the neural network and $\mathcal{X}$ is the instance space. Golowich et al. [91] further improved bounds of this form to

$$\widehat{\mathrm{RAD}}_N(\mathcal{N}) \leq \frac{1}{\sqrt{N}} \left(\sqrt{2 \log(2) L} + 1 \right) \left(\prod_{j=1}^{L} \|W_j\|_F \right) \left(\max_{x \in \mathcal{X}} \|x\| \right) \tag{3.4}$$

3.1.3 Rank and margin-based complexity measures

Despite its (marginal) impact on generalisation in practice through explicit penalisation, these bounds fail to explain some simple behaviours of neural networks. Multi-layer perceptrons (MLPs) and even convolutional neural networks (without special structures like skip connections) show the interesting property of positive homogeneity. A function f is k-positive homogenous if there exists an integer k such that for any positive constant c and an element x in the domain of f, we have $f(cx) = c^k(x)$. It is easy to verify that a k-layer MLP is k-positive homogenous in the space of parameters. An important consequence of this fact is that, for a k-layer MLP, multiplying all the weights of the network with a positive constant does not alter the test error of the model even though it increases the norm of the weights. Thus, using a suitably large positive number and the positive homogeneity of MLPs, the complexity measures in Theorem C can be increased to any arbitrarily large number without altering the generalisation error of the model, which is a direct contradiction to what a generalisation bound is supposed to achieve. This shows a simple failure case of norm-based complexity measures like in Theorem C. Margin-based measures, discussed below, can be used to overcome this particular

[3] Theorem C uses Empirical Rademacher Complexity while the generalisation bound in Theorem B is with Rademacher complexities. However, one can be transferred to the other. See `https://www.cs.ox.ac.uk/people/james.worrell/rademacher.pdf` for further information.

failure case.

Before progressing further we will first define the concept of margin. Consider a hypothesis class $\mathcal{H}$ and the learning task to be multi-class classification. Every hypotheses $h \in \mathcal{H}$ takes a vector $\mathbf{x}$ from $\mathcal{X} \subset \mathbb{R}^d$ and generates a probability distribution over the classes identified by the k dimensions of $\mathcal{Y} \subseteq \Delta^k$, the k-dimensional simplex. The standard strategy for prediction using this model is to output the index of $h(\mathbf{x})$ which has the largest magnitude i.e. $\mathbf{argmax}_i\, h(\mathbf{x})[i]$. In definitions 10 and 11, we define the concept of margin and margin loss respectively.

Definition 10 (Margin)**.** *The margin of a hypothesis h at a data sample $(\mathbf{x}, y)$ is defined as*

$$\gamma(h; \mathbf{x}, y) = h(\mathbf{x})[y] - \max_{j \neq y} h(\mathbf{x})[j]$$

where $h(\mathbf{x})[j]$ indexes the j^{th} element of $h(\mathbf{x})$.

Definition 11 (Margin Loss)**.** *The margin loss is defined as*

$$\ell_{\gamma^*}(h; \mathbf{x}, y) = \begin{cases} 0 & \gamma(h; \mathbf{x}, y) \geq \gamma^* \\ 1 & \gamma(h; \mathbf{x}, y) < \gamma^* \end{cases} \tag{3.5}$$

where γ is the margin defined in Definition 10. We will use $\widehat{\mathrm{R}}_{\gamma,N}(h)$ and $\mathcal{R}_\gamma(h)$ to represent the empirical and the expected margin risk, respectively, of the classifier h. The normal classification risk is simply $\mathcal{R}_0(h)$.

Spectrally normalized margin bounds Consider an MLP $g \in \mathcal{N}$ as defined in Definition 1. Let the network g be parameterised by a sequence of weight matrices $\mathbf{W}_1, \cdots, \mathbf{W}_L$ with l_k neurons in the k^{th} layer. The width of the network W is the maximum of $\{l_1, \ldots, l_L\}$.

Definition 12 (Spectral Complexity from Bartlett et al. [22], Neyshabur et al. [196])**.** *The* spectral complexity *R_g of such a network $g \in \mathcal{N}$ is defined as*[4]

$$R_g = \left(\prod_{i=1}^{L} \|\mathbf{W_i}\|_2 \right) \left(\sum_{i=1}^{L} \frac{\left\|\mathbf{W_i}^\top\right\|_{2,1}^{2/3}}{\|\mathbf{W_i}\|_2^{2/3}} \right)^{3/2} \tag{3.6}$$

[4]For simplicity, we assume 1-Lipschitz Activation functions

where $\|\cdot\|_{p,q}$ *is the entry-wise* p, q *norm and is defined in Equation* (A.4) *along with other basic linear algebra concepts.*

The following theorem provides a generalisation bound for neural networks g whose weight matrices $\{\mathbf{W}_1, \cdots, \mathbf{W}_L\}$ have bounded spectral complexity R_g[5].

Theorem D (Spectrally Normalized Margin Bounds from Bartlett et al. [22])**.** *For any MLP* $g \in \mathcal{N}$ *with width* W *and whose weight matrices* $\{\mathbf{W}_1, \cdots, \mathbf{W}_L\}$ *have bounded spectral complexity* R_g*, any* $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ *drawn i.i.d. from a distribution over* $\mathbb{R}^d \times \{1, \ldots, k\}$*, and for any* $\gamma > 0$ *, the following holds with probability at least* $1 - \delta$*:*

$$\mathcal{R}(g) \leq \widehat{\mathrm{R}}_{\gamma,N}(g) + \tilde{\mathcal{O}}\left(\frac{\sqrt{\sum_{i=1}^{N} \|\mathbf{x}_\mathrm{i}\|^2} R_g}{\gamma N} \log W + \sqrt{\frac{\log \frac{1}{\delta}}{N}}\right) \tag{3.8}$$

Though the bound in Theorem D does not have a direct exponential dependence on the depth of the network L, Neyshabur et al. [196] points out that for any matrix $\mathbf{W}_i$,

$$\frac{\left\|\mathbf{W}_\mathrm{i}^\top\right\|_{2,1}}{\|\mathbf{W}_\mathrm{i}\|_2} \geq 1.$$

Thus if $\max_{i \leq N} \|\mathbf{x}_\mathrm{i}\| = R_{\mathcal{X}}$ and R_g is the spectral complexity as defined above, then the excess error term in Theorem D can be written as

$$\frac{\sqrt{\sum_{i=1}^{N} \|\mathbf{x}_\mathrm{i}\|^2} R_g}{N} = \tilde{\mathcal{O}}\left(\left(R_{\mathcal{X}} \prod_{i=1}^{L} \|\mathbf{W}_\mathrm{i}\|_2\right) \sqrt{\frac{L^3}{N}}\right). \tag{3.9}$$

The expression in Equation (3.9) still depends exponentially on L if the spectral norm of the individual matrices $\mathbf{W}_i$ is not bounded below one. Even if the norms are bounded, the bound becomes trivial when $L \geq N^{\frac{1}{3}}$. In contrast, Golowich et al. [91] shows a bound, which under suitable bounds on various norms, is independent

[5]The authors [22] also defines a collection of *reference matrices* $(\mathbf{M}_1, \ldots, \mathbf{M}_L)$ with the same dimensions as $\mathbf{W}_1, \ldots, \mathbf{W}_L$. This can be adjusted for various network structures to get the best bound. Then, the spectral complexity is defined as

$$R_g = \left(\prod_{i=1}^{L} \|\mathbf{W}_\mathbf{i}\|\right) \left(\sum_{i=1}^{L} \frac{\left\|\mathbf{W}_\mathrm{i}^\top - \mathbf{M}_\mathrm{i}^\top\right\|_{2,1}^{2/3}}{\|\mathbf{W}_\mathrm{i}\|_2^{2/3}}\right)^{3/2}. \tag{3.7}$$

of the size of the network.

Neyshabur et al. [196] develop a slightly different expression for spectral complexity through a Pac-bayesian analysis. The original paper gives a qualitative comparison between the two bounds. Their form of spectral complexity can be written as follows where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm and $\|\cdot\|_2$ is the 2-operator norm (see Equation (A.1)).

$$R_g = \left(\prod_{i=1}^{L} \|\mathbf{W_i}\|_2^2 \sum_{i=1}^{L} \frac{\left\|\mathbf{W}_\mathrm{i}^\top\right\|_{\mathrm{F}}^2}{\|\mathbf{W}_\mathrm{i}\|_2^2} \right)^{\frac{1}{2}} \tag{3.10}$$

The term $\frac{\left\|\mathbf{W}_\mathrm{i}^\top\right\|_{\mathrm{F}}^2}{\|\mathbf{W}_\mathrm{i}\|_2^2}$ in Equation (3.10) is also known as the stable rank of matrices. In Chapter 4, we develop an algorithm to directly control the spectral complexity (see Equation (3.10)) of neural networks during training. Our experiments on a wide range of settings show that this indeed helps generalisation in practice.

3.1.4 Complexity measures based on compression

Next, we discuss a slightly different idea for formulating generalisation bounds of neural networks. The main idea in Arora et al. [16] is that if the performance of a *large* network on the training set can be copied very closely by a *small* network, then one can use the low empirical risk of the *large* network to guarantee that the smaller network will also have a low empirical risk. Then, standard complexity based generalisation bounds can leverage the low complexity of the *small* network to guarantee a better generalisation guarantee for the small network. As the small network behaves similar to the large network, this generalisation guarantee can then be transferred to the large network. To be able to use results of this kind in practice, we need to a) obtain a large network that gets small training error and b) show that the large network can be closely emulated by a smaller network. We will refer to the copying of the large network by a small network as *compressing* the network. The first is easy to obtain in practice as neural networks often train to very small training error. For the second, Arora et al. [16] defines a set of sufficient properties for the neural network to be compressible. We list these properties below and then evaluate them in later chapters to show that neural networks obtained via our regularisations

enjoy some of these desirable properties.

Definition 13 defines (γ, S)-compressible functions. A function h is (γ, S)-compressible with respect to a set of functions $\mathcal{G}$ if one of the functions in G can simulate h within an error tolerance of γ. Then, assuming that $\mathcal{G}$ is a set of low-complexity functions and h has a low empirical error on a fixed dataset, Theorem E guarantees that the function from $\mathcal{G}$ that closely simulates h will have a low expected risk.

Definition 13 ((γ,S)-compressible using helper string s (see Arora et al. [16]))**.** *Let $\mathcal{A}$ be the set of all possible parameters and $G_{\mathcal{A},s} = \{g_{A,s} | A \in \mathcal{A}\}$ be a class of classifiers indexed by trainable parameters A and a fixed string s. Then, for $\gamma > 0$, a classifier h is (γ, S)-compressible with respect to $G_{\mathcal{A},s}$ using helper string s if there exists $A \in \mathcal{A}$ such that for any $x \in S$ and all y.*

$$|h(\mathbf{x})[y] - g_{A,s}(\mathbf{x})[y]| \leq \gamma.$$

Theorem E. *Suppose $G_{\mathcal{A},s} = \{g_{A,s} | A \in \mathcal{A}\}$ where A is a set of q parameters each of which can have at most r discrete values and s is a helper string. Let S be a training set with N samples. If the trained classifier f is (γ_c, S)-compressible via $G_{\mathcal{A},s}$ with helper string s, then there exists $A \in \mathcal{A}$ such that $\forall \gamma \geq 2\gamma_c$ with probability at-least $1 - \delta$ over the training set,*

$$L_0(g_A) - \hat{L}_\gamma(f) \leq \sqrt{\frac{1}{2N}\left(q \log r + \log \frac{1}{\delta}\right)} = \tilde{\mathcal{O}}\left(\sqrt{\frac{q \log r}{m}}\right).$$

The problem with Theorem E is that it does not provide a generalisation guarantee for the original classifier but only for the compressed version. Below, we will identify some properties of neural networks which will allow us to circumvent this problem. These properties have been presented in Arora et al. [16].

Compression based bounds and data dependent properties Let $\mathcal{S}$ be a set of N examples drawn i.i.d. from any probability distribution on $\mathbb{R}^d \times \{1 \cdots k\}$ and let g be an MLP as described above with L layers. In particular, let $\mathbf{W}_i$ be the

weight matrix of the linear transformation of the i^{th} layer, let $\mathbf{z}_i$ be the pre-activation representation of the i^{th} layer, and let ϕ be the activation function. For any two layers $i \leq j$, denote by $M^{i,j}$ the operator for composition of these layers and by $J_{\mathbf{x}}^{i,j}$ the Jacobian of this operator at input $\mathbf{x}$. Then, the following data-dependent properties of the network are defined for a given network. We empirically evaluate these properties later in the thesis to understand the impact of our regularisations on the behaviour of the network and how they affects properties like noise-sensitivity and generalisation.

Definition 14 (Layer Cushion)**.** *For any layer i, the layer cushion is defined as the largest number μ_i such that for any $x \in S$:*

$$\mu_i \|\mathbf{W}_\mathrm{i}\|_F \|\phi(\mathbf{z}_{\mathrm{i}-1})\| \leq \|\mathbf{W}_\mathrm{i}\phi(\mathbf{z}_{\mathrm{i}-1})\|.$$

Definition 15 (Inter-Layer Cushion)**.** *For any two layers $i \leq j$, the inter-layer cushion is defined as the largest number $\mu_{i,j}$ such that for any $x \in S$:*

$$\mu_{i,j} \left\|\mathbf{J}_{\mathbf{z}_\mathrm{i}}^{\mathrm{i,j}}\right\|_F \|\mathbf{z}_\mathrm{i}\| \leq \left\|\mathbf{J}_{\mathbf{z}_\mathrm{i}}^{\mathrm{i,j}} \mathbf{z}_\mathrm{i}\right\|.$$

Furthermore, define the minimal inter-layer cushion $\mu_{i\rightarrow}$ as $\min_{i\leq j\leq d} \mu_{i,j} = \min\{1/\sqrt{l_i}, \min_{i<j\leq d} \mu_{i,j}\}$.

Definition 16 (Activation Contraction)**.** *The activation contraction is defined as the smallest number c such that for any layer i and any $x \in S$,*

$$\|\mathbf{z}_\mathrm{i}\| \leq c\|\phi(\mathbf{z}_\mathrm{i})\|.$$

Theorem F (Bounds Based on Compression, [16])**.** *For any MLP g as defined above*[6]*, any probability $0 < \delta \leq 1$, and any margin γ, Algorithm 1 in Arora et al. [16] generates $\widehat{g}$ such that with probability at least $1-\delta$ over the training set and the*

[6]The result also requires some additional assumption on a property known as Inter-Layer Smoothness (c.f. Definition 7 in Arora et al. [16])

randomness in creating $\widehat{g}$, the following holds

$$\mathcal{R}(\hat{g}) \leq \widehat{\mathrm{R}}_{\gamma,N}(g) + \tilde{\mathcal{O}}\left(\sqrt{\frac{c^2 L^2 \mathcal{R}_{\mathcal{X}}^2 \sum_{i=1}^{L} \frac{1}{\mu_i^2 \mu_{i\rightarrow}^2}}{\gamma^2 N}}\right),$$

where c, μ_i, ρ_d, and $\mu_{i\rightarrow}$ are defined above, and L is the number of layers. This $\widehat{g}$ can be thought of as the compressed version of g.

Through these properties, this bound captures more data-dependent characteristics than the other generalisation bounds discussed above. The sample complexity obtained using Theorem F for standard neural networks is also more realistic than any of the other results discussed in the earlier paragraphs. This suggests that we need to find structures in the data and training algorithm that play a role in determining the generalisability of neural networks trained on real-world data as opposed to just considering the architecture of the neural network. One way of doing this is to identify what properties of the learned network reflect said properties of the data and the algorithm. In Chapter 4, we look at *Empirical Lipschitzness* to measure the sensitivity of the network on the dataset. In Chapter 6, we look at layer cushion (see Definition 14) to measure the noise-sensitivity of neural networks, and in Chapter 7, we look at the entropy of the predicted distribution over target class labels to understand causes of mis-calibration of neural networks. We find that these data-dependent properties of neural networks are a better indicator of the network's behaviour than data-agnostic properties.

3.2 Adversarial Robustness of Neural Networks

In a benign real world setting, where the probability distributions from which the training and the testing data are sampled are identical, generalisation, as discussed in Chapter 2, provides guarantees on the reliability of deployed machine learning systems. However, machine learning algorithms are also used in settings where the assumption that the data distribution remains stationary between training and deployment does not hold. In such settings, a direct use of generalisation theories, as discussed in the last section, does not provide meaningful guarantees of the reliability

of the machine learning system.

This can be particularly worrisome when a malicious adversary has the power to change the data distribution for their own benefit. Adversarial attacks [29, 248] — where an adversary imperceptibly changes the data to force the model into making a mistake, is one example of such a setting. Recently, this has been shown to be extremely relevant for neural networks [62, 29, 248, 93, 43, 205, 186]. This section will discuss the threat model, different types of adversaries, ways to measure vulnerability against these adversaries, and existing work in the literature regarding how to protect from them.

3.2.1 Characterising an adversary

Before formally defining adversarial error, we need to first characterise what it means for the adversary to a) *imperceptibly change the data* and b) *be successful in forcing the model in making a mistake.*

Threat model The first part is captured by defining a *threat model* - which puts a constraint on what the adversary is allowed to do while perturbing the data. There are different types of constraints the adversary can enforce:

1. Computational Constraint – A computational constraint restricts the number of operations the adversary is allowed to execute while perturbing the data point. For example, if an adversary is allowed to do only k steps of gradient ascent while constructing the perturbed data point, that is an example of a computational constraint on the adversary.

2. Information Theoretic Constraint – An information theoretic constraint restricts the adversary to ensure that the information contained in the perturbed data is not too different from that in the original data. For example, an ℓ_p bounded adversary ensures that the ℓ_p norm of the induced perturbation is small.

3. Knowledge Constraint – The adversary can also be constrained by the information it has access to while constructing the attack. Some adversaries might have full knowledge of the model they are attacking whereas others may not

and have to construct their proxy of the model while crafting the attack. Examples of this distinction can be seen between white box and black box attacks respectively.

Formally, the threat model of the adversary can be defined as a function $\mathcal{A} : \mathcal{X} \to \mathcal{P}^{\mathcal{X}}$ where $\mathcal{P}^{\mathcal{X}}$ denotes the power set of $\mathcal{X}$. $\mathcal{A}$ is a function that maps a point in $\mathcal{X}$ to a set of points in $\mathcal{X}$ that can be feasibly obtained by the constrained adversary. For an ℓ_p bounded adversary with radius r, $\mathcal{A}(\mathbf{x})$ is the set of all points within an ℓ_p ball of radius r around $\mathbf{x}$.

$$\mathcal{A}(\mathbf{x}) = \left\{ \mathbf{z} : \|\mathbf{z} - \mathbf{x}\|_p \leq r \right\}$$

Remark 1. *For real application purposes, "imperceptibly changing the data" usually means to be imperceptible to a human. The definition of the threat model above (a function from a single example to a set of examples) is powerful enough to represent this. However, it is difficult to provide a precise mathematical definition of such a threat model. Thus, research in adversarial robustness has stuck to easy-to-define and computationally tractable threat models. However, neither does this mean that more powerful threat models do not exist nor does it mean that this simple threat model is of no practical utility.*

The second component of characterizing an adversary is defining what it means to be successful in forcing the model to make a mistake. Generally, this corresponds to forcing the value of the loss function of the attacked model to increase on the perturbed data point as compared to the original data point. Usually, adversarial robustness is discussed in the context of supervised classification problems. Forcing the model to make a mistake in this context corresponds to forcing the model to make a classification error on the adversarially perturbed data point despite being correct on the original data point.

Combining these two components, the job of an adversary is to find a perturbed data point within its threat model that maximally increases the loss value of the

perturbed data point. Given an adversary $\mathcal{A} : \mathcal{X} \to \mathcal{P}^{\mathcal{X}}$, a learned classification model $h : \mathcal{X} \to \mathcal{Y}$, a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, and an *original data point* $(\mathbf{x}_d, y)$, the *adversarially perturbed data point* $\mathbf{x}_a$ is the solution of the following optimisation problem.

$$\mathbf{x}_a = \max_{\mathbf{x} \in \mathcal{A}(\mathbf{x}_d)} \ell\left(h\left(\mathbf{x}\right), y\right) \tag{3.11}$$

Even for relatively simple threat models like ℓ_p bounded adversaries, the maximisation problem can be difficult to solve exactly. Especially in neural networks, the classifier h is a non-convex function and thus the inner maximisation problem is non-convex. Thus, attacks are usually created by considering various approximations to the problem. The most common approach to solve this is projected gradient ascent.

Definition 17 (ℓ_p Adversarial Error)**.** *For any distribution $\mathcal{D}$ defined over $(\mathbf{x}, y) \in \mathbb{R}^d \times \mathcal{Y}$, any classifier $h : \mathbb{R}^d \to \mathcal{Y}$, and any $\gamma > 0$, the* γ-adversarial *error is*

$$\mathcal{R}_{\mathrm{Adv},\gamma}(h; \mathcal{D}) = \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\exists \mathbf{z} \in \mathcal{B}_\gamma\left(\mathbf{x}\right); h\left(\mathbf{z}\right) \neq y\right], \tag{3.12}$$

where $\mathcal{B}_\gamma^p\left(\mathbf{x}\right)$ is the ℓ_p ball of radius $\gamma \geq 0$ around $\mathbf{x}$ under the ℓ_p norm.

In the adversarial robustness literature, the adversarial error is usually defined by Definition 17. This particular definition measures the risk when the adversary is only constrained by information theoretic constraints, more specifically an ℓ_p norm ball. Most work has looked at ℓ_∞ [93, 172] constraints though some have considered ℓ_2 [43], ℓ_1 [48], and ℓ_0 [43] balls as well. However, when reporting empirical results, computational constraints are also necessarily included for practical reasons. Thus most practical evaluations of adversarial robustness use a combination of computational, information theoretic, and knowledge constraints. Głuch and Urbanke [100] discuss a generalisation of the knowledge constraint where the adversary is characterised by the number of queries it is allowed to make to the attacked model while constructing the attack. If this number is unbounded, then the adversary is called a white box adversary and if the number is zero, the adversary is called a black box

adversary. Any finite non-zero number characterises the strength of the adversary.

We will now discuss some commonly used adversaries, which we will also use later in Chapters 5 and 6 and then discuss some commonly used adversarial defence methods.

3.2.2 Adversarial attacks

One of the first instances of adversarial attacks with Neural Networks is in the work of Goodfellow et al. [93] where the adversary is constrained both computationally and information theoretically, in terms of ℓ_p bounds. When the perturbed data point is constrained to lie within an ℓ_p norm of a certain radius, we will refer to the radius as the perturbation budget of the adversary. The adversary in Goodfellow et al. [93] is allowed to do one step of gradient ascent while ensuring that the perturbed point is within an ℓ_∞ perturbation budget of the original data point. Here, the magnitude of the perturbation budget defines the strength of the adversary. Known as Fast Sign Gradient Method (FGSM), this adversary is characterised by the perturbation budget ϵ and attacks a point $\mathbf{x}_d$ by generating the adversarially perturbed $\mathbf{x}_a$ as

$$\mathbf{x}_a = \mathbf{x}_d + \epsilon \cdot \operatorname{sign}\left(\nabla_x \ell\left(h\left(\mathbf{x}_d\right), y\right)\right). \tag{3.13}$$

This is in fact one step of projected gradient descent on the cross-entropy loss function ℓ where the projection set is an ℓ_∞ ball of radius ϵ around $\mathbf{x}_d$. The other important characteristic of this attack is that the sole aim of the adversary is to force h to misclassify $\mathbf{x}_a$ without any specific target for what $\mathbf{x}_a$ should be misclassified to.

Multi-step adversaries Equation (3.13) and the FGSM adversary describe an attack threat model where the adversary is constrained to only one step of projected gradient ascent; a stronger version of this allows the adversary is allowed to take $T \geq 1$ steps of length α [152] while satisfying the constraint of being within the ℓ_∞ perturbation budget. Algorithm 1 describes this process. In this case, the definition of the threat model includes a computational constraint hyper-parameter T to control the number of allowed gradient ascent steps, a hyper-parameter α for

the length of each gradient step, and an information theoretic parameter ϵ for the perturbation budget. We will refer to this as the Iter-FGSM adversary. It is one of the most commonly used threat models as it allows varying the computational and the information theoretic constraints independently. It has been used in a series of works including Madry et al. [172] and Zhang et al. [294].

Algorithm 1 Multi-Step Fast Gradient Sign Method

input Original Data $(\mathbf{x}_d, y)$, classification model (h), loss function (ℓ), Threat model (T, ϵ, α)

1: $\mathbf{x}_a^0 \leftarrow \mathbf{x}_d$
2: **for** $t \in \{0, \cdots T-1\}$ **do**
3: $\quad \mathbf{x}_a^{t+1} \leftarrow \text{Clip}_{\mathbf{x}_d,\epsilon}\left(\mathbf{x}_a^t + \alpha \nabla_x \ell\left(h\left(\mathbf{x}_a^t\right), y\right)\right)$
4: **end for**

output $\mathbf{x}_a^T$

The Clip operation in Algorithm 1 is specific to the ℓ_∞ bounded adversary and image data where each pixel ranges between 0 and 255 and is defined as

$$\text{Clip}_{x,\epsilon}(z) = \min\left(255, x+\epsilon, \max\left(0, x-\epsilon, z\right)\right)$$

However, these attacks can also be extended to norms other than the ℓ_∞ norm by changing the projection operator to a different operator depending on the norm. For example, Szegedy et al. [248] used the ℓ_2 norm.

Minimum-Norm unbounded attacks Equation (3.14) describes a generic form of such attacks where $\|\cdot\|$ is a particular norm and $h(\mathbf{x}) \neq h(\mathbf{x}+\mathbf{r})$ can be cast into our generic adversarial example formulation of Equation (3.11) by considering the loss function ℓ to be the classification error function $\ell_{01}(y, \hat{y}) = \mathbb{I}\{y \neq \hat{y}\}$. Another form of attack where there is no specific computational or information-theoretic constraints are minimum norm unbounded attacks.

$$\begin{aligned} &\min_{\mathbf{r}\in\mathbb{R}^d} \|\mathbf{r}\| \\ &\text{s.t. } h(\mathbf{x}) \neq h(\mathbf{x}+\mathbf{r}). \end{aligned} \tag{3.14}$$

A specific instantiation of this with the ℓ_2 norm is the DeepFool adversary by Moosavi-Dezfooli et al. [186]. However, measuring the success of these unbounded

attacks is different from measuring the success of bounded attacks as, by design, these unbounded attacks are more likely to succeed. Thus Moosavi-Dezfooli et al. [186] measure the ratio $\frac{\|\mathrm{r}\|}{\|\mathbf{x}\|}$ for successful misclassifications, which indicates how far the attack had to move the original data point for the classification algorithm to fail.

Targeted attacks The attacks we saw so far are known as *untargeted attacks*; the sole aim of the adversary is to make the target model misclassify the data point without any constraint on what it should be misclassified as. In a k-class multiclass classification problem, this means that the adversary is satisfied if the model classifies the perturbed data point into one of the $k-1$ incorrect classes. A *Targeted adversarial attack* changes the data point imperceptibly with the aim that the classifier model classifies the perturbed data point into a label of the adversary's choosing.

In addition to the arguments of Algorithm 1, a targeted adversarial attack adversary would also usually take a target label y_t. The only change in the algorithm would be the update step, where instead of increasing the loss value for the correct label, the adversary would now decrease the loss value for the target label as follows:

$$\mathbf{x}_a^{t+1} \leftarrow \mathrm{Clip}_{\mathbf{x}_d,\epsilon}\left(\mathbf{x}_a^t - \alpha \nabla_x \ell\left(h\left(\mathbf{x}_a^t\right), y_t\right)\right).$$

Kurakin et al. [152] observed that for datasets with a large number of classes and varying degrees of significance in the difference between classes, untargeted adversarial attacks can result in *uninteresting* misclassifications, such as mistaking one breed of sled dog for another breed of sled dog. Thus, in order to create visually striking adversarial mis-classifications, they developed a method which they refer to as *Iteratively Least Likely Class Method* (Iter-LL-FGSM). Their adversary forces the classifier to classify the perturbed data point to a label that would have been the least likely class for the original data point. For a given data point $\mathbf{x}$ and a classifier $h : \mathcal{X} \to \mathcal{Y}$, they choose the least likely class as y_{LL} as

$$y_{\mathrm{LL}} = \underset{y \in \mathcal{Y}}{\mathbf{argmax}}\, \ell\left(h\left(\mathbf{x}\right), y\right).$$

The algorithm for the Iteratively Least Likely Class Method is described in Algorithm 2.

Algorithm 2 Iterative Least Likely Class Method

input Original Data $(\mathbf{x}_d, y)$, classification model(h, ℓ), Threat model (T, ϵ, α)

1: $\mathbf{x}_a^0 \leftarrow \mathbf{x}_d$
2: **for** $t \in \{0, \cdots T-1\}$ **do**
3: $\quad y_{\text{LL}} = \mathbf{argmax}_{y \in \mathcal{Y}} \ell\left(h\left(\mathbf{x}\right), y\right)$
4: $\quad \mathbf{x}_a^{t+1} \leftarrow \text{Clip}_{\mathbf{x}_d, \epsilon}\left(\mathbf{x}_a^t - \alpha \nabla_x \ell\left(h\left(\mathbf{x}_a^t\right), y_{\text{LL}}\right)\right)$
5: **end for**

output $\mathbf{x}_a^T$

Unstability of fixed steps adversarial attacks Both Iter-FGSM and Iter-LL-FGSM run for a fixed number of steps T. An adaptive version of these attacks add the adversarial noise for a maximum of T steps but stops early upon successful misclassification even before T steps. We show empirical evidence that an attack that adds noise for a fixed number of steps [153, 152] to the input is significantly weaker than one that stops on successful misclassification. While it would be natural to expect that once a classifier has misclassified an example, adding more adversarial perturbation will only preserve the misclassification, Figure 3.1 suggests that a misclassified example can be possibly classified correctly upon further addition of noise.

(a) Iter-LL-FGSM (b) Iter-FGSM

Figure 3.1: An adversarial example that has successfully fooled the classifier in a previous step can be classified correctly upon adding more perturbation. Figure 3.1(a) and 3.1(b) refers to the two attack schemes - Iter-LL-FGSM and Iter-FGSM respectively.

Let $y_a(\mathbf{x}; k)$ be the label given to $\mathbf{x}$ after adding adversarial perturbation to $\mathbf{x}$ for k steps. We define *instantaneous accuracy* $(a_{\mathcal{I}}(k))$ and *cumulative accuracy* $(a_{\mathcal{C}}(k))$

as

$$a_{\mathcal{I}}(k) = 1 - \frac{1}{N}\sum_{i=1}^{N} \mathcal{I}_{0,1}\left\{y_a(\mathbf{x};k) \neq y_a(\mathbf{x};0)\right\} \tag{3.15}$$

$$a_{\mathcal{C}}(k) = 1 - \frac{1}{N}\sum_{i=1}^{N} \max_{1\leq j\leq k}\left\{\mathcal{I}_{0,1}\left\{y_a(\mathbf{x};j) \neq y_a(\mathbf{x};0)\right\}\right\}. \tag{3.16}$$

In Figure 3.1, we see the *instantaneous accuracy* and the *cumulative accuracy* for a ResNet model trained on the CIFAR10 dataset (see Appendix B for a description of the model and the dataset) where $\alpha = 0.01, \epsilon = 0.1$, and T is plotted in the x-axis. The cumulative accuracy is by definition a non-increasing sequence. However, surprisingly the instantaneous accuracy is not monotonic and has a lower rate of decrease than the cumulative accuracy. It also appears to stabilise at a value much higher than the cumulative accuracy.

This is by no means an exhaustive list of proposed adversarial attacks but merely a brief list of the main categories of these attacks. We discuss them because we use them in later chapters to measure adversarial vulnerability. Similarly, the defences proposed in the next section are not an exhaustive list of all defence approaches that have been proposed against adversarial attacks.

3.2.3 Adversarial defences

There are various types of defences that have been proposed in the literature. Some of them are based on the regularisation of the neural network [55], some on data-augmentation [172, 294], and others on theoretical guarantees [157, 56].

Regularisation-based defences One of the main causes of adversarial vulnerability in neural networks is their high sensitivity to input perturbations in certain directions. This means that when the input is slightly perturbed in that direction there is an unexpectedly large change in the output of the neural network. Sensitivity of the network is commonly measured with various measures of lipschitzness[7]. Multiple works have discussed constraining these measures of lipschitzness of neural networks in the context of adversarial vulnerability. The product of operator

[7]We discuss lipschitzness in greater detail in Section 4.2.2

norms of linear transformations of the individual layers of neural networks is one way of measuring lipschitzness of neural networks. Szegedy et al. [248] discuss how linear and convolution layers, whose operator norms are greater than one, magnify the adversarial perturbation as the perturbation propagates through the layers. If the operator norms are smaller than one then the transformations will attenuate the magnitude of the perturbation as it propagates through the layers and protect against high sensitivity in neural networks. To obtain this in practice, Cisse et al. [55] implement *Parseval tight frames*, which are extensions of orthogonal matrices to non-square matrices. By constraining the parameters of linear and convolutional layers to remain in Parseval tight frames, they guarantee that the lipschitzness of these layers will be smaller than one. While Cisse et al. [55] and Szegedy et al. [248] use properties of individual weight matrices to calculate the lipschitzness of the entire network, Tsuzuku et al. [260] use zeroth-order information of the network to approximate the lipschitzness. Tsuzuku et al. [260] introduce lipschitz margin training, where they directly regularise a differentiable approximation of the lipschitzness of the network. Another way of approximating the lipschitzness is through first-order information obtained via the Jacobian of the network. Multiple works [222, 221, 130, 171] use Jacobian regularisation to constrain the lipschitzness and thus the sensitivity of the neural networks. All of these approaches based on regularising the lipschitzness of neural networks provide empirical boosts to adversarial robustness.

However, such regularisations pose a problem with regards to the faithful measurement of the true robustness of neural networks. As pointed out by Athalye et al. [18], directly penalizing gradients around points from the training data sets up gradient-based attacks to fail without regularising the loss surface outside the immediate vicinity of that point. These approaches protect against the computation of successful gradient-based adversarial attacks around the training points but the network still remains vulnerable to other attacks, for example, attacks that do not use first-order information at training points. This phenomenon is called "Obfuscated Gradients" and Athalye et al. [18] show that multiple defences that exhibit this phenomenon can be overcome with specially designed attacks.

Noisy training-based defence Another way of regularising a neural network is to add noise to the training procedure. Bishop [30] show that training with noise is, sometimes, equivalent to a form of Tikhonov regularisation. Jin et al. [133] show that adding stochastic noise to the input and model parameters during training improves the adversarial robustness of convolutional neural network (CNN) models. Dhillon et al. [68] propose randomly pruning activations and their experiments indicate that this strategy improves the robustness of networks without requiring further fine-tuning. Sankaranarayanan et al. [228] perturb intermediate layer activations and use this as a regulariser during training to impart robustness to trained networks.

Defence based on smooth training In regularisation-based defences, we saw that regularising a network to constrain the operator norm of each layer improves robustness of the network in practice. The reason why the operator norm of different layers increases in practice is a combination of positive homogeneity of neural networks and exponential loss functions like cross-entropy with one-hot target labels. Positive homogeneity means that multiplying the weights by a positive constant does not alter the accuracy of the network but can change the logits in the penultimate layer. A key property of exponential loss functions with one-hot target labels is that the only way to reduce the loss to zero is by increasing the magnitude of weights and consequently the logits to infinity. Thus, neural network training is inherently biased towards magnifying the operator norms of each layer of the network, and this leads to heightened sensitivity and worse robustness of neural networks.

A natural defence to control this behaviour is to prevent the exponential loss function from encouraging the magnification of the operator norms of the weight matrices in a neural network. One way to do this is by replacing the one-hot target vectors with smoothed vectors. Two common techniques in the literature to achieve this are *defensive distillation* [116] and *label smoothing* [249]. In defensive distillation, a neural network is first trained on the classification problem, and then the one-hot target label vectors in the training dataset are replaced with class probability vectors obtained from the trained network's prediction on the points from the training dataset. Then a new network is trained on the modified dataset (where the

label vectors are smoothed). First proposed in Hinton et al. [116] as a method for transferring knowledge from larger networks to smaller networks, Papernot et al. [206] adapted this strategy for imparting adversarial robustness and renamed it *defensive distillation*.

Another form of regularisation that directly prevents models from increasing the operator norms of their layers is through *label smoothing* [249]. It is a regularisation technique that introduces noise for the labels. For a small constant ϵ, in a k-class classification problem, label smoothing regularises a model by replacing the hard 0 and 1 classification targets in the one-hot target label vector with $\frac{\epsilon}{k-1}$ and $1-\epsilon$ respectively. Its benefits were initially observed for calibration in Müller et al. [189]. Goibert and Dohmatob [89] conducted a more systematic study in the context of adversarial robustness and explored various smoothing techniques especially those that are more suited for adversarial robustness.

Data-augmentation-based defences The most popular form of adversarial defence by far is *adversarial training* [172] and its more sophisticated variants [294]. Essentially, it exploits the fact that adversarial examples are so ubiquitous in deep neural networks that constructing them is relatively straightforward using gradient-based attacks as discussed in the previous section. Adversarial training augments the training procedure by replacing the original training point with an adversarial training point constructed by an adversary. It can be viewed as minimizing an approximate minimisation of the adversarial risk defined in Definition 17.

However, it has been observed empirically that adversarial training causes a perceived tradeoff between robustness and accuracy [153]. Zhang et al. [294] provide a differentiable upper bound on the combined adversarial and natural test error and term the algorithm for minimizing this upper bound TRADES. Using TRADES instead of adversarial training, they show that this perceived trade-off can be avoided to a certain extent. This is perhaps the most commonly used variant of adversarial training in practice.

Defences based on detection Another line of defence is based on detecting adversarial examples so that the classifier is not forced to predict a label for an example

if a detector can detect the example to be an adversarial example. Hendrycks and Gimpel [111] present three methods to detect adversarial examples which rely on properties like identifying the subspaces in the image space exploited by the adversary and abnormal softmax values for adversarial examples as compared to natural examples. Yang et al. [282] use temporal dependency in audio signals to detect adversarial examples specifically for Automated Speech Recognition tasks. In a k-class classification problem, Yin et al. [286] use k different detectors to do the detection. In particular, if the model predicts the class to be i, the i^{th} detector is used to confirm that the image is not adversarially perturbed. Akhtar et al. [10] learn both a detector network and a Perturbation Rectification Network (PRN) and use the PRN to rectify the perturbation when a perturbation is detected by the detector network.

However, such detection methods usually suffer a great deal from incorrect and weak evaluations. Carlini and Wagner [44] and Tramer et al. [255] show how a dozen of these detection methods can be successfully bypassed by designing better adversaries. This suggests that a lot of these properties that were thought to be inherent to adversarial examples are in fact not inherent to the problem of adversarial robustness but rather artefacts of the particular techniques used for constructing the adversaries in their evaluation techniques. This calls for developing defence methods with theoretical guarantees for robustness agnostic to the particular technique used for generating adversarial attacks during empirical evaluation.

Defence with guarantees The famous Goodhart's law [94] states that

> Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes. (Goodhart)

Arguably, the cat and dog race that has prevailed in the realm of adversarial attacks and defences can be attributed to this law. As discussed, the minimisation of the true adversarial risk is a difficult problem. Thus most defence techniques tend to minimise one particular approximation of the risk. Future attacks that break the defence usually succeed by breaking that assumption inherent to that approximation.

However, recent work has looked at creating provable defences against adversarial vulnerability. In terms of guarantees of robustness, the most powerful of those come from the verification literature. Verification, in the context of adversarial robustness, means to provide a theoretical guarantee that if inputs to the neural network belong to a pre-defined set of inputs (eg. inputs within a norm ball), then the outputs of the network satisfy a desired property (eg. the logit corresponding to the correct label has a higher value than all the other logits). Research from the formal verification community has used Satisfiability Modulo Theory (SMT) solvers to provide guarantees against bounded norm perturbation attacks [76, 122, 137]. However, the verification bounds from these approaches, while sufficient to guarantee robustness, ends up guaranteeing robustness against only very weak adversaries. Thus, they might be of little practical significance.

To make these bounds more useful, a line of research has tried to alter the training process of neural networks so that the bounds obtained from these trained networks are more practically relevant [213, 276]. Another line of research has proposed approaches that use *branch and bound* algorithms [41, 52, 253] to provide stronger bounds. However, all of these methods are usually hard to adapt for large networks as the computational complexity of these approaches depends directly on the SMT problem instances. Moreover, they are usually limited to problems with piecewise linear activation functions like ReLU and maxpooling and it has proved hard to adapt them to general architectures and activation functions unless further approximations are made. Consequently, this is an active direction of future research with a potentially significant impact on certifying the robustness of deep neural networks.

Another approach to providing guaranteed robustness is through the approach of PixelDP (*Pixel Differential Privacy*) [157, 159] or randomised smoothing [56]. While verification techniques, as discussed in the previous paragraph, prove the robustness of an existing classifier, randomised smoothing techniques construct a new classifier that is smooth within a *certified radius*; the *certified radius* of the smoothed classifier depends on a hyper-parameter of the smoothing process and

properties of the original model.

Specifically, consider a classification problem from $\mathbb{R}^d$ to the set of labels $\mathcal{Y}$ and a base classifier $f : \mathbb{R}^d \to \mathcal{Y}$ that has been trained to do the classification. The objective of randomised smoothing is to use a smoothing distribution $\mathcal{N}_S$ to obtain a smoothed classifier $g : \mathbb{R}^d \to \mathcal{Y}$ from f, such that when queried at $\mathbf{x} \in \mathbb{R}^d$, the smoothed classifier g returns whichever class f is most likely to return when $\mathbf{x}$ is perturbed by noise drawn from the smoothing distribution $\mathcal{N}_S$.

The larger the variance of the smoothing distribution, the greater the magnitude of the radius that can be certified for the smoothed classifier. One of the key questions of research in this direction is what kind of smoothing distribution should be used for a particular kind of adversary. Lecuyer et al. [157] and Cohen et al. [56] used the gaussian distribution to certify against an adversary with an ℓ_2 perturbation bound. Yang et al. [279] generalised this to other kinds of adversaries and proposed a generic technique to find the best possible distribution for multiple ℓ_p perturbation bounds. Awasthi et al. [19] further improved these approaches by leveraging natural low-rank representations of data to provide improved guarantees.

It is easy to see that if the variance of the smoothing distribution is very large, then the classifier may become a constant predictor. Mohapatra et al. [183] shows that with increasing noise variance, the decision regions shrink in size and the classifier becomes overwhelmingly likely to always predict one specific class and ignore the others. Thus, there seems to be a tradeoff between robustness and accuracy when randomised smoothing is used as a technique to guarantee robustness.

3.2.4 Tradeoffs associated with robustness

Several works have suggested that robustness is inherently at odds with other measures that are important in the learning problem or even unavoidable, entirely.

Fawzi et al. [79] constructs a distribution where data $\mathcal{X}$ is generated from a generative model $\mathcal{X} = g(\mathbf{r})$, where $\mathbf{r} \in \mathbb{R}^d$ is sampled from an isotropic gaussian and is, roughly, of euclidean norm $\sqrt{d}$. They show that due to isoperimetry of gaussian distribution, no classifier is robust to adversarial perturbations of euclidean norm $\mathcal{O}(1)$ irrespective of its (clean) expected error (Definition 2). When g is an

L-lipschitz function, this corresponds to perturbations of at most $\mathcal{O}(L)$. Tsipras et al. [259] show a slightly more optimistic setting, where robust classification is possible but not simultaneously achievable with low expected error. In particular, they construct a simple setting where, for any classifier, as expected error approaches 0%, adversarial error (Definition 17) tends towards 100%. They use a distinction between *robust* and *non-robust* features, where the classifier needs to use the non-robust features to attain a low expected error at the cost of a large adversarial error. Similarly, using the robust features leads to low adversarial error at the cost of large expected error. Zhang et al. [294] show another simple setting where the bayes-optimal classifier obtains a low expected error but a large adversarial error whereas the optimal adversarial error is obtained by a constant classifier at the cost of a large expected error.

Interestingly, in both of these examples, neither additional data nor choosing a different representation of data can help in making the classifiers more robust. However, humans are considered to be robust as well as accurate classifiers at least for natural images. Thus, any result that rejects the possibility of a robust and accurate classifier entirely is highly unlikely to apply to real-world data. Schmidt et al. [234] consider a setting where learning with low adversarial error is possible without being at odds with expected error but requires more data than learning a classifier with low expected error. In their setting, learning a classifier with low expected error requires just one sample whereas learning a classifier with low adversarial error requires $\Omega\left(\sqrt{d}\right)$ samples where d is the dimensionality of the input space i.e. a polynomial separation in sample complexity. Bubeck et al. [40] show that this gap is tight, in the sense that if non-robust learning (low expected error) is possible with polynomial sample complexity then robust learning is also possible with polynomial sample complexity. While this might seem like an optimistic result, empirical evidence suggests that this is not observed in practice.

Bubeck et al. [39] and Degwekar et al. [64] showed (under some cryptographic assumptions) that there exist learning tasks where a computationally efficient robust and accurate classifier exists, can be learnt from a small number of samples, but the

learning algorithm will necessarily be computationally expensive. A computationally efficient classifier is defined as a classifier that can compute the output, given the input in time polynomial in the size of the input. Degwekar et al. [64] further extend along this direction showing settings where robust classification is possible but only via a computationally expensive classifier. Their results show that, for certain distributions, computationally simple hypothesis classes do not admit a robust classifier whereas robust learning is possible using a different, more complex (computationally) hypothesis class. Madry et al. [172] provide some empirical validation of the hypothesis in Degwekar et al. [64] in the sense that choosing wider and deeper neural networks have generally led to lower adversarial error even when simpler models (shallower or narrower neural networks) models suffice for low expected risk.

Montasser et al. [185] establish that there are hypothesis classes with finite VC dimensions i.e. are *properly* PAC-learnable but are only *improperly* robustly PAC learnable. This implies that to learn the problem with small adversarial error, a different concept class is required whereas for low expected risk, the original hypothesis class suffices. Unlike Degwekar et al. [64], the difference in complexity between the two concept classes here is not that of computational complexity but rather statistical (or sample) complexity.

All of these works indicate that for different distributions learning tasks robustness might be at odds with different quantities — accuracy, statistical complexity, computational complexity of learning, and computational complexity of the hypothesis class. However, real-world data distributions and learning tasks might not possess the same hardness as the distributions and learning tasks used in the construction of these examples. This presents a hope that these tradeoffs, despite appearing in practice, are not inherent to the real world learning problems but are only artefacts of the algorithms and hypothesis classes we use in practice.

3.3 Calibration of Neural Networks

Calibration in multi-component systems Generalisability of machine learning models, as discussed in Section 3.1, allows us to extrapolate our confidence in the performance of a machine learning model from a fixed-size training set to *unseen*

data. Adversarial robustness of machine learning (Section 3.2) models provides confidence that the performance of the model will not be vulnerable to malicious changes to the input data. Another component of the reliability of machine learning models is *calibration*. When machine learning models are deployed in the real world, they are often deployed to assist humans in certain tasks or they are used in an ensemble with other machine learning models.

For example, in medical science, machine learning models are used to combine multiple medical signals to determine rapidly whether the patient might require imminent critical care [54, 142, 36] or to diagnose health conditions based on imaging data [201, 14]. However, these solutions are usually meant to assist a clinician as opposed to remove the clinician from the decision-making loop. Thus, the clinician needs to have an estimate of the confidence of the machine learning system on its prediction. Such an estimate allows the clinician to reliably trust the machine learning system's diagnosis or overrule its diagnosis with their own diagnosis.

In automated driving systems, machine learning is used for tasks like detecting lane changes, labelling obstacles, steering, accelerating, and braking. Multiple machine learning components, usually relying on different sensors, are used to make these decisions. To reliably combine the predictions of these different components, a measure of the confidence of the individual components is essential. For example, when visibility is poor it is safer to trust a component that does not depend on image data whereas when information on the road is presented with signs and visual cues, it is safer to trust a vision-based component. Therefore, if a system can calibrate its confidence on its various components, their outputs can be combined effectively to provide a reliable prediction.

In both of these situations, an estimation of the uncertainty of the decisions made by the individual machine learning models is required for the safe deployment of the entire system. One particular measure of this uncertainty is through the notion of calibration. Deep neural networks are designed to output a vector in a $|\mathcal{Y}|$-dimensional simplex. This is usually done by first producing a k-dimensional vector $\mathbf{z} \in \mathbb{R}^k$ and then a softmax function transforms $\mathbf{z}$ element-wise to $\text{softmax}(\mathbf{z}) \in \Delta^k$.

$$\text{softmax}\left(\mathbf{z}\right)_i = \frac{\exp\left(\mathbf{z}_i\right)}{\sum_{i=1}^{k}\exp\left(\mathbf{z}_i\right)} \tag{3.17}$$

This vector can be interpreted as a conditional likelihood distribution $\mathbb{P}\left[\mathcal{Y}|\mathcal{X},h\right]$[8] over the label space $\mathcal{Y}$ given the input $\mathcal{X}$ and the model (architecture-cum-parameters) h. A calibrated model returns a *probability* vector where the individual entries $\mathbb{P}\left[\mathcal{Y}=y|\mathcal{X}=\mathbf{x},h\right]$ of the vector are truly indicative of the actual likelihood of the class y as a correct prediction for the input $\mathbf{x}$. In the rest of this section, we discuss the different notions of calibrations discussed in the literature, different approaches for measuring calibration of models in practice, and finally approaches to calibrate a model.

3.3.1 Types of calibration

A strong notion of calibration is discussed in Kull et al. [147], Widmann et al. [271], Vaicenavicius et al. [261], and Kumar et al. [148]. Let $h\left(\mathbf{x}\right)\left[y\right]$ be the conditional probability that the model h attributes to the y^{th} class on seeing the input $\mathbf{x}$. Strong calibration requires that for a fully calibrated model h, the following holds for all $y\in\mathcal{Y}$ and all $p\in[0,1]$:

$$\mathbb{P}\left[\mathcal{Y}=y\ |h\left(\mathbf{x}\right)\left[y\right]=p\right]=p. \tag{3.18}$$

This implies that for any class y, for all examples to which the model assigns a conditional likelihood of p for belonging to that class, the model should be correct on a p fraction of those examples. This is an especially strong notion of calibration as it accounts for all classes for every example even when it is not the correct class for that example.

A relatively weaker notion of calibration, known as *weak calibration*, is discussed in Guo et al. [99]. This notion only requires that the conditional probability for the predicted class be calibrated as opposed to all the classes being calibrated. This notion of calibration requires that for all $p_{max}\in[0,1]$, we have

[8]This is a slight abuse of notation as we have not defined h to be a random variable.

$$\mathbb{P}\left[\mathcal{Y} = \underset{y}{\mathbf{argmax}}\, h\left(\mathbf{x}\right)[y] \,\middle|\, \max_{y} h\left(\mathbf{x}\right)[y] = p_{max}\right] = p_{max}. \tag{3.19}$$

Kull et al. [147] also consider another measure, known as *class-specific calibration*, that is not as restrictive as strong calibration but is stronger than weak calibration. Under this measure, calibration is enforced only for a specific class. Class-specific calibration is ensured for the class c^* if the following holds for all values $p_{c^*} \in [0, 1]$:

$$\mathbb{P}\left[\mathcal{Y} = c^* \,\middle|\, h\left(\mathbf{x}\right)[c^*] = p_{c^*}\right] = p_{c^*}. \tag{3.20}$$

3.3.2 Measuring miscalibration

The previous section discusses three notions of calibration expressed as conditional probabilities. However, the conditional probabilities cannot be computed accurately with a finite number of samples since the conditional likelihood is a continuous random variable. The natural way of dealing with issues like these in practice is through discretisation of the continuous random variable. Ideally, weak calibration, as defined in Equation (3.19), would be measured by grouping all examples $\mathbf{x}$ whose likelihood of the predicted class is exactly p and then measuring the absolute difference between the accuracy of the model in that group and the value p. If the difference is zero for all values of p, then the model is perfectly weakly-calibrated. However, as mentioned before this is impractical as the value p is continuous. So, the problem is usually tackled through discretisation of the $[0, 1]$ interval for values of p.

Expected calibration error One of the most commonly used metrics to measure calibration in practice is the Expected Calibration Error (ECE) [191]. The measure is characterised by an integer hyper-parameter $M > 1$ that is used for discretisation. The $[0, 1]$ interval is divided into M equal-width bins $\{B_1, \ldots, B_M\}$ where $B_i = \left[\frac{i-1}{M}, \frac{i}{M}\right]$. Then, all predictions on the test-set are categorised into one of these bins depending on the conditional likelihood (i.e. the softmax value) of the predicted class. For example, if an example is predicted to be in class c with probability p,

that example is placed in the bin B_i such that $\frac{i-1}{M} \leq p < \frac{i}{M}$. Then, the confidence C_i and accuracy A_i of bin B_i are computed as the average conditional likelihoods and the average accuracy respectively of the examples in that bin. ECE is measured as

$$\mathrm{ECE}\,(h) = \sum_{i=1}^{M} \frac{|B_i|}{N} \left|A_i - C_i\right|, \tag{3.21}$$

where $|B_i|$ represents the number of examples in the bin B_i and N is the total number of examples in the test-set.

Maximum Calibration Error While ECE can be thought of as the L_1 calibration error, another popular measure of calibration error is L_∞ calibration error, otherwise known as the Maximum Calibration Error (MCE) [191]. To compute MCE, the bins B_i, their corresponding accuracies A_i, and confidences C_i are computed as above. Then, MCE is computed as

$$\mathrm{MCE}\,(h) = \max_i \left|A_i - C_i\right|. \tag{3.22}$$

Modifications to the calibration errors Other versions, including the L_2 version [148], have also been used in literature to measure the calibration error. A consequence of discretising p uniformly is that each bin can end up with a wildly varying number of samples. For neural networks, most examples end up in a bin with a high value of p. Thus, the low confidence bins consist of very few samples and this can heavily impact the measures of both ECE and MCE. To overcome this, both Nixon et al. [198] and our work (discussed in Chapter 7) propose the use of adaptive binning strategies (Adaptive ECE or AdaECE) where the bins are created to ensure that every bin is equally populated.

Classwise expected calibration error Another drawback of the calibration errors is that they are designed to only measure weak calibration as they account for the confidence of only the predicted class. Stronger definitions of calibration require that all the classes be calibrated. Nixon et al. [198] propose a new metric called the Static Calibration Error (SCE) to overcome this. We will refer to this as classwise ECE instead. To compute this metric, M separate bins indexed as

$B_{i,j}$ are created for each of the K classes. The bin $B_{i,j}$ represents the i^{th} bin for the j^{th} class. Once all the examples have been binned into their respective bins, the accuracies $A_{i,j}$ and confidences $C_{i,j}$ are computed for each bin. Then, classwise ECE is measured as follows

$$\text{Classwise-ECE}\,(h) = \frac{1}{K}\sum_{i=1}^{M}\sum_{j=1}^{K}\frac{|B_{i,j}|}{N}\,|A_{i,j} - C_{i,j}| \tag{3.23}$$

Reliability plots These metrics of calibration error summarise the error into a single statistic without offering an insight into which of the bins were more mis-calibrated and which were less. A useful visual tool for this is the reliability plot introduced in Niculescu-Mizil and Caruana [197]. The reliability plot is a bar plot where the x-axis measures the confidence and the y-axis measures the accuracy. Each bar in the reliability plot represents a bin and is put on the X-axis in increasing order of the average confidence of that bin. The height of the bar denotes the average accuracy of the examples in that bin. For a fully calibrated model, the height of each bar should be the same as its X-axis label. If the height is more than its X-axis label, then the model is said to be under-confident and if the height is less than the X-axis label, then the model is said to be over-confident.

Other metrics of calibration While the ECE, its variants, and reliability plots have remained the most widely used metrics for mis-calibration, several other loss functions are also used to measure mis-calibration of models. The most popular of these are the Negative Loss Likelihood and the Brier Score [37]. While some works have looked at minimizing these measures directly to boost both accuracy and calibration, those techniques have also led to an apparent trade-off between accuracy and calibration. It is easy to see that a model can be extremely inaccurate while being fully calibrated and sometimes, this solution is preferred over the more desirable one of being nearly accurate with very low calibration error [99].

3.3.3 Approaches for calibrating deep neural networks

Multiple approaches have been proposed in the literature for calibrating a neural network in practice. They can be broadly categorised into approaches for post-hoc

calibration, and approaches for calibration during training.

Post-hoc calibration The basic idea for post-hoc calibration is to learn a function that maps the uncalibrated output of the machine learning model into calibrated likelihoods. Different approaches for post-hoc calibration vary in the nature of this function. Without any restriction on the nature of the function, the function itself can overfit to the training data and not generalise to new data. Thus various strategies of post-hoc calibration restrict this function to different classes of parametric and non-parametric functions.

Platt scaling Platt [211] proposed a post-hoc calibration technique through scaling the output of the model via a sigmoid function parameterised by two learnable scalars a and b. In this approach, the output of the machine learning model $h(\mathbf{x})$ is replaced by $\frac{1}{1+\exp(ah(\mathbf{x})+b)}$. Their initial approach was applied for SVMs and was motivated by the empirical observation that a sigmoid function captured the relationship between SVM scores and empirical conditional likelihoods for many commonly used datasets .

Guo et al. [99] modified Platt scaling for use in neural networks by removing the bias term in the affine transformation and extending it to multi-class classification. They termed this approach *temperature scaling*, which operates on the pre-softmax vector (also referred to as the logit vector) by replacing the original softmax function Equation (3.17) with Equation (3.24)

$$\text{softmax}_{\text{TS}}(\mathbf{z};T)_i = \frac{\exp(\mathbf{z}_i/T)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i/T)} \tag{3.24}$$

This approach, however, suffers from multiple drawbacks. For example, while it scales the logits to reduce the network's confidence in incorrect predictions, it also reduces the network's confidence in predictions that are correct [149]. Moreover, it has been observed that temperature scaling does not calibrate a model under data distribution shift [202]. Despite these drawbacks, temperature scaling is perhaps the most commonly used method for post-hoc calibration. This is due to its simplicity and impressive performance on a wide range of neural network architectures [99].

However, when the exact nature of the mapping function is unknown it makes little sense to use a sigmoid function with a fixed structure. For this purpose, Zadrozny and Elkan [288] proposed a non-parametric technique referred to as binning. In this technique, the outputs of the function on the training set are sorted in decreasing order of magnitude of the conditional likelihood and then placed into bins of equal size. When a test example is evaluated, it is first placed into one of the bins depending on the output of the model on that example. Then the output of the model on that test example is replaced with the average accuracy of training examples in that bin.

Another approach that is mid-way between a parametric model and a fully non-parametric method is *isotonic regression* [200], which is a form of non-parametric regression. In isotonic regression, the learned function is chosen from the class of all non-decreasing (or isotonic) functions. The underlying intuition for why this is suitable for calibration is that even though the base machine learning classifier might not output the correct conditional likelihoods, it should still rank the classes properly. In that case, the correct mapping from the space of model outputs to the true conditional likelihood is an isotonic function.

Several other post-hoc calibration techniques have been proposed in recent literature. These include further modifications to temperature scaling like Bin-wise Temperature Scaling (BTS) [131], learning sample-wise temperature parameters [69], and Dirichlet Calibration [147].

Calibrating during training While post-hoc calibration techniques are easy to implement and can be used to calibrate a model without interfering in its training process, there are multiple issues associated with these techniques. First, post-hoc methods require a considerably large validation set to tune themselves on. Second, there is an extra computational overhead associated with training the calibration technique once the training of the base model is complete. Methods that calibrate the model during training overcome these issues by producing a calibrated model at the end of training without requiring an extra calibration phase of training. One popular way of calibrating during training is by training a network using modified

loss functions.

The brier Score, introduced by Brier [37], is a loss function that measures the accuracy of probabilistic predictions. It can intuitively be thought of as a squared error between the predicted likelihood vector and the one-hot target. Let the K-dimensional one-hot representation of the label y be y^{01}. Then, the brier score of a model h on the example $(\mathbf{x}, y)$ is

$$BS\left(h; \mathbf{x}, y\right) = \sum_{i=1}^{N} \sum_{j=1}^{K} \left(h\left(\mathbf{x}_i\right)[j] - y_i^{01}[j]\right)^2. \tag{3.25}$$

The Brier score can be algebraically decomposed into two components associated with accuracy and calibration. Thus, minimizing Brier loss provides a simultaneously accurate and calibrated model as we observe in Chapter 7. However, we also observe that while minimizing the Brier score provides better calibration error than NLL, it still trades off calibration error for test error.

We could try to minimise the calibration errors we saw in the previous sections directly, but most of the errors like ECE and MCE are non-differentiable metrics. Kumar et al. [149] propose a differentiable proxy for the calibration error, called *Maximum Mean Calibration Error (MMCE)*, which they add as an extra regulariser during training in addition to the negative log-likelihood loss.

Several other approaches have been proposed for calibration during training. Müller et al. [189] adapt label smoothing for calibration for reasons similar to those discussed in the context of adversarial robustness. Similarly, Thulasidasan et al. [252] show that using mixup [295] training also leads to a drop in calibration error.

In Chapter 7, we propose the use of focal loss [165] as a loss function that can simultaneously provide high accuracy and low calibration errors. We show that minimizing focal loss is approximately equivalent to minimizing a regularised Bregman divergence where the regularisation component helps with calibration and the Bregman divergence is associated with the usual negative log-likelihood classification loss. Our experiments show that our method performs better than the other methods mentioned in this section. Combining focal loss with temperature scaling further boosts its performance. We use the various metrics discussed in this section

to report the performance.

Model ensembles and other approaches An approach that combines post-hoc calibration and calibration during training are model ensembles. Model ensembles combine the predictions of multiple models, trained on different subsets of the data, and they have long been known to improve generalisation [106] of machine learning models. The diversity of the models in an ensemble have been known to help in generalisation and recent works [240, 140] have developed approaches to increase this diversity in practice. Raftery et al. [212] and Stickland and Murray [244] have shown that ensemble diversity also helps with improved model calibration. Zhong and Kwok [297] uses ensembles of SVMs, logistic regressors, and boosted decision trees for improved calibration.

In the case of neural networks, various works including Lakshminarayanan et al. [154] and Ovadia et al. [202] have shown that ensemble of neural networks can also help with calibration in deep learning. Calibration of neural networks is closely related to estimating predictive uncertainty, which is something bayesian neural networks (eg. Variational inference or MCMC techniques) excel at. However, bayesian neural networks are computationally intensive and hard to implement. In contrast, *deep ensembles* proposed by Lakshminarayanan et al. [154] is a simple approach that scales well and uses a combination of ensembles and proper scoring functions. Ashukha et al. [17] provides empirical evidence that deep ensembles outperform some bayesian neural networks approaches in regards to calibration. Wilson and Izmailov [274] develops an approach called MultiSWAG which combines the benefits of bayesian deep learning and deep ensembles to propose an approach that is not computationally more expensive than deep ensembles (at train time) but provides better calibration than deep ensembles. However, their approach is indeed more expensive to use at test time. Wenzel et al. [270] provides empirical evidence that the posterior predictive induced by the Bayes posterior yields systematically worse predictions compared to point estimates obtained from SGD. Interestingly, we have not found papers that propose ensemble techniques and bayesian neural networks for calibration to also report results on calibration metrics like ECE and MCE, as

we discussed above. Nevertheless, all of these approaches are complementary to the usage of a different loss function, which is the approach we take in this thesis.

3.4 Privacy in deep learning

So far we have discussed notions of reliability in machine learning viz. generalisation, robustness, and calibration that deal directly with guaranteeing certain properties in the learned machine learning models. In this section, we will look at a different issue that arises when machine learning models are deployed as a service (MLaaS). With rapidly evolving machine learning algorithms for a wide range of tasks, growing computational capabilities to deploy these algorithms in practice, and ever-growing access to the internet and online services, MLaaS is poised to become, if it is not already, a ubiquitous component of our lives. However, along with its numerous benefits, this also poses major potential issues arising from inadequate security protocols to protect the data that is fed into these machine learning algorithms either during training, during inference, or both.

While privacy issues arising from amassing and storing data for training a machine learning model are important, in this thesis we will not focus on that aspect of privacy in machine learning. Instead, we will focus on an important recent paradigm called *prediction as a service*, whereby a service provider with expertise and resources can make predictions on data provided by the clients. However, this approach requires trust between the service provider and client; there are several instances where clients may be unwilling or unable to provide data to service providers due to privacy concerns. Examples include assisting in medical diagnoses [144, 31], detecting fraud from personal finance data [87], and detecting online communities from user data [81]. The ability of a service provider to predict on private data without being able to see the actual data can alleviate concerns of data leakage. We will look at a paradigm where a service provider has trained a machine learning model and customers can use that model by sending their data to the service provider.

3.4.1 Techniques for Privacy-Preserving Inference

Consider a medical diagnosis company that provides a service where customers can upload chest radiographs and the company's algorithms diagnose whether the patient is suffering from a bone fracture. If the algorithms are accurate and reliable under the notions of reliability that we have seen in the previous sections, this can be a very valuable service with the ability to rapidly diagnose multiple patients without human intervention. However, without sufficient trust between the company and the customer, the customer might be disinclined to upload their chest radiographs to the service. For example, a dishonest company can use the images to estimate their general lung health and sell that information to insurance companies or predict whether the customer smokes and sell that information to tobacco companies. Both of these outcomes can have very damaging financial and health consequences for the customer. To prevent these possibilities, the machine learning algorithms should protect the privacy of the user while making accurate predictions. The broad field concerned with this is commonly referred to as *Privacy-Preserving Inference*. The following paragraphs discuss three generic techniques for tackling this problem.

Trusted Execution Environments Broadly, the problems discussed above arise due to the execution of a machine learning algorithm on private data in an untrusted environment. Trusted Execution Environments (TEE) like Intel SGX [174], ARM TrustZone [11], and Sanctum [59] present a solution to this problem by providing a secure environment to run the code. TEEs are a secure area within the main processor of a computer. They are an isolated environment that runs in parallel with the operating system and guarantees that code and data loaded inside is protected with respect to integrity and confidentiality. TEEs use hardware and software protections to guarantee that any code and data stored in that secure enclave can be accessed only by code in that enclave irrespective of privileges in the software stack. Therefore, even the OS or the hypervisor cannot access the information stored in the enclave.

However, there are significant computational and security challenges associated with this approach. First, the secure enclave is usually extremely limited in memory

and computation. Large models cannot be loaded on it due to memory limitations and computations using large model require significant paging, which introduces computational overheads. Second, these enclaves do not have a large number of parallel threads, which further slows down the computation. Finally, the enclave is maintained by a *trusted* third party. If the third party is dishonest, the third party can use its admin privileges on the system to mount side-channel attacks on the enclave (see Section 6 in Hunt et al. [124]).

Multi-Party Computation (MPC) Another way of approaching the problem is by designing a protocol that itself emulates the secure third party, which collects the data from multiple parties, evaluates the function, and returns the result to all (or a specific set of) parties. This allows two or more parties to evaluate a function without disclosing their data to one another or anyone else. Garbled Circuits [283] and the GMW protocol [90] are examples of protocols that aim to solve this problem. They look at the following scenario. There are n players each having a piece of secret information $\{x_1, \ldots, x_n\}$ and there is a boolean function g that needs to be evaluated on these n inputs. However, the function needs to be evaluated without any of the players learning anything more from the process than they would have learned from just observing the output $g(x_1, \ldots, x_n)$.

While this seems to provide the required security guarantees, there are multiple issues associated with this technique especially if the protocol is used in the Prediction As A Service framework. First, the protocol assumes a level of honesty among the players which might not hold in the real world. In particular, it requires that the players stick to the given protocol. Ideally, we would like a protocol that protects against a malicious adversary. Second, MPC requires multiple rounds of communication between the players and it requires all the parties to execute some components of the computation. In the Prediction As A Service framework, customers might have low capacity devices and be unable to execute complex computations on their device, and depending on their internet access, multiple rounds of communication can be prohibitively expensive or time-consuming. Ideally, we would like a protocol where the customer operates once on the data at the beginning and sends it to the

service provider and then receives the output. Third, all parties in the protocol need to have access to the function being evaluated. Thus, if one o the players is the service provider and the other is the user, it requires the user to have access to the (possibly encrypted) machine learning model on their private device so that they can do some of the evaluation on their private device. This is often undesirable as service providers would be unwilling to share their models with customers. In addition, the service providers would have to share a new model every time they update their model.

Homomorphic Encryption While TEEs and MPC schemes provide some level of privacy and reliability in using Prediction As A Service, these approaches protect against weaker threat models than we would like. TEEs do not provide cryptographic privacy guarantees in the execution environment, thus their computations are vulnerable to side-channel attacks and rely on the third party being honest. MPC schemes also rely on partial honesty among the parties and require multiple rounds of communication between the service provider and the user. Further, MPCs require the function that is going to be evaluated to be shared between the different users. Ideally, we would like to develop a protocol that can overcome all of these disadvantages.

One particular way to achieve this is if the data provided by the customer is cryptographically hidden from the machine learning model while still enabling the model to make accurate, albeit encrypted, predictions. This protection is exactly what "Encrypted Prediction As A Service (EPAAS)" defined by us in Chapter 8 provides. The basic service required by EPAAS is that the service provider has access to a learned machine learning model in plaintext. The customer encrypts the private personal data and sends the encrypted data to the service provider, along with the public key but not the private key. The service provider computes an encrypted prediction on the received data using their model and sends the encrypted prediction back to the customer, who then decrypts it. The framework of *Fully Homomorphic Encryption (FHE)* is ideal for this paradigm. *Homomorphic encryption*, first proposed by Rivest et al. [219], is an encryption methodology that allows

certain operations on it without decrypting it first. Gentry [84] proposed the first FHE scheme that allows performing arbitrarily many operations on the encrypted data. Since then several other schemes have been proposed [85, 86, 35, 72, 53].

3.4.2 Challenges in using Homomorphic Encryption

While EPAAS using homomorphic encryption schemes offer powerful privacy protections, the major challenge associated with using homomorphic encryption is its computational inefficiency. Without significant changes to the machine learning model and improved algorithmic tools, homomorphic encryption does not scale to modern deep neural networks.

Indeed, already there have been several recent works trying to accelerate predictions of machine learning models on fully homomorphically encrypted data. In general, the approach has been to approximate all or parts of a machine learning model to accommodate the restrictions of an FHE framework. Often, certain kind of FHE schemes is preferred because they allow for "batched" parallel encrypted computations, called SIMD operations [241]. This technique is exemplified by the CryptoNets model [88]. While these models allow for high-throughput (via SIMD), they are not particularly suited for the Prediction As A Service framework for individual users, as single predictions are slow. Further, because they employ a leveled homomorphic encryption scheme, they are unable to perform many nested multiplications, a requirement for state-of-the-art deep learning models [110, 120].

In Chapter 8, we will look at a novel solution that demonstrates how existing work on Binary Neural Networks (BNNs) [139, 61] can be adapted to produce efficient and highly accurate predictions on encrypted data. We show that a recent FHE encryption scheme [53] which only supports operations on binary data can be leveraged to compute all of the operations of BNNs. To do so, we develop specialised circuits for fully-connected, convolutional, and batch normalization layers [125]. Additionally, we design tricks to sparsify encrypted computation that reduces computation time even further. We lay down some important computational and privacy criteria that need to be satisfied by an EPAAS framework and we discuss why most recent approaches fail them. Then we discuss various types

of Homomorphic Encryption schemes and why our particular choice of encryption scheme is suitable for our method.

4. Improving Generalization via Stable Rank Normalization

In Sections 2.1 and 3.1, we discussed the importance of complexity measures and regularisation techniques for certifying generalisation in machine learning models. This chapter looks at how one particular complexity measure derived from the study of generalisation in neural networks can be explicitly penalised, as a regulariser, to improve generalisation in practice. We leverage recent results on the generalisation of deep networks to yield a practical low-cost method to normalize the weights within a network using a scheme, we call Stable Rank Normalization (SRN).

Remark 2 (Normalization And Regularisation)**.** *Both* Normalization *and* regularisation *refer to techniques for controlling the complexity of neural networks. The main difference between the two is that while* regularisation *refers to a Tikhonov style regularisation and softly penalises the complexity,* normalization *usually implements an Ivanov style regularisation and guarantees that a hard constraint on the complexity of the network is satisfied.*

In addition, normalization techniques for neural networks also ensure that the desired property is satisfied throughout training. Hence, by changing the loss landscape [229], normalization techniques also have a significant impact on optimisation. Some commonly used normalization techniques are Batch normalization [126] that controls the mean and variance of the activations in one batch of examples, Spectral Normalization [182] that controls the spectral norms of individual layers, and Weight Normalization [226] that controls the euclidean norms of the weight parameters.

The motivation for SRN comes from the generalisation bound for neural networks (NNs) given by Neyshabur et al. [196] and Bartlett et al. [22] and discussed previously in Theorem D (in particular its variant discussed in Equation (3.10)). Recall that the excess error due to Equation (3.10) scales as $\mathcal{O}\left(\sqrt{\prod_i^L \|\mathbf{W}_\mathrm{i}\|_2^2 \sum_{i=1}^L \mathrm{srank}(\mathbf{W}_i)}\right)$ where L and $\|\mathbf{W}\|_2$ represents the number of layers and the spectral norm of the i-th linear layer $\mathbf{W}_i$, respectively. It depends on two parameter-dependent quantities:

1. The product of scale-dependent spectral norms of individual layers $\prod_i^L \|\mathbf{W}\|_2$, also referred to as the Lipschitz constant upper-bound and

2. The sum of scale-independent *stable ranks* of each layer $\sum_{i=1}^L \text{srank}(\mathbf{W}_i)$. Stable rank is a softer version of the rank operator and is defined as the squared ratio of the Frobenius norm to the spectral norm.

We refer to the spectral norm as a scale-dependent term as, like all norms, the spectral norm of a matrix is *absolutely homogenous* and increases proportionally with the scaling of the matrix. On the other hand, stable rank is a ratio of two norms, and thus upon multiplying the matrix with a scalar, the scaling of the two individual norms cancels each other. Like the rank of a matrix, stable rank is thus insensitive to the scaling of a matrix.

It is common to regularise a complexity bound obtained from generalisation bounds to observe better empirical generalisation in practice. Some examples, relevant for neural networks, include weight decay for L_2 regularisation, Path-SGD [194, 296] for Path-norm regularisation, and margin-jacobian regularisation [269] for controlling a data-dependent generalisation bound proposed in Wei and Ma [269]. However, the empirical impact of simultaneously controlling both, the spectral norm and the stable rank, on the generalisation behaviour of NNs has not been explored yet possibly because of the difficulties associated with optimizing stable rank. This is precisely the goal of this chapter. Based on extensive experiments across a wide variety of NN architectures, we observe that controlling them simultaneously indeed improves the generalisation behaviour of NNs. We also observe improved training of Generative Adversarial Networks (Generative Adversarial Networkss (GANs)) [92] with our technique.

To this end, we propose Stable Rank Normalization (SRN) which allows us to simultaneously control the Lipschitz constant and the stable rank of a linear operator. Note that the widely used Spectral Normalization (SN) [182] allows explicit control over the Lipschitz constant, however, as we discuss later in this chapter, it is neither the optimal solution to the spectral normalization problem and neither does it have any impact on the stable rank. Unlike SN, the SRN solution, to controlling

the stable rank, is optimal and unique despite being a non-convex problem. It is one of those rare cases where an optimal solution to a provably non-convex problem can be easily computed. Computationally, the proposed SRN technique for NNs is no more complicated than SN, just requiring the computation of the largest singular value, which can be done efficiently using the power iteration method [180].

4.1 Main contributions

Before going into a detailed discussion of the problem, in this section, we briefly state the main contributions of this chapter. We divide the list of contributions into two sections- a) theoretical and algorithmic contributions and b) Experimental Results.

Theoretical and Algorithmic Contributions

1. We propose Stable Rank Normalization (SRN)— a novel normalization scheme for simultaneously controlling the Lipschitz constant and the stable rank of a linear operator.

2. We also present a unique and optimal solution to the provably non-convex stable rank normalization problem.

3. Finally, we devise and efficient and easy algorithm to implement SRN for NNs.

Experimental Results Although SRN is in principle applicable to any problem involving a sequence of affine transformations, considering recent interests, we show its effectiveness when applied to the linear layers of deep neural networks. We also experiment with GANs and show that SRN prefers learning discriminators with low empirical Lipschitz while providing improved Inception, FID and Neural Divergence scores [98].

1. We perform extensive experiments on a wide variety of NN architectures (DenseNet, WideResNet, ResNet, Alexnet, and VGG) for the analyses and show that SRN improves classification accuracy on a wide variety of architectures.

2. While providing the best classification accuracy (compared against standard training, vanilla training, and SN training), neural networks trained with SRN shows remarkably less memorisation, even on settings that are known to be hard to generalise in.

3. Further, networks trained with SRN show much smaller sample complexity measured using complexity measures proposed in recent works.

4. Applying SRN to Generative Adversarial Networks (GANs) [92] improves the performance of GANs measured via various metrics and makes them more resilient to memorisation.

We also note that although SN is widely used for training GANs, its effect on the generalisation behaviour over a wide variety of multi-class classification neural networks has not yet been explored. To the best of our knowledge, we are the first to do so.

4.2 Definitions and preliminaries

Neural Networks Consider a neural network $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ parameterised by $\theta \in \mathbb{R}^p$, each layer of which consists of a linear mapping followed by a non-linear[1] activation function. Recall that it is defined as follows in Definition 1.

Definition 1 (Multi-Layer Perceptron)**.** *A Multi-Layer Perceptron (MLP) is a hierarchial model with a sequence of L layers. The i^{th} layer is parameterized by a matrix $\mathbf{W}_i \in \mathbb{R}^{\ell_{i-1} \times \ell_i}$ where ℓ_i represents the width of the i^{th} layer and $W = \max_{1 \leq L} \ell_i$ is the width of the MLP. The hypothesis class of MLPs $\mathcal{M}$ is*

$$\mathcal{M} : \{h_\theta \mid h_\theta(\mathbf{x}) = \mathbf{W}_1 \phi_1 (\cdots \phi_{l-1} (\mathbf{W}_{L-1} \phi_L (\mathbf{W}_L \mathbf{x}))), \theta = \{\mathbf{W}_1, \cdots, \mathbf{W}_L\}\}$$

Each ϕ_i is an activation function and can all be identical or different depending on the network configuration. In practice, they are usually the same. When ϕ operates on a vector, it operates element-wise on each element of the vector.

[1] *e.g.* ReLU, tanh, sigmoid, and maxout.

For classification tasks, given a dataset with input-output pairs denoted as $(\mathbf{x} \in \mathbb{R}^d, \mathbf{y} \in \{0,1\}^k; \sum_j y_j = 1)$ [2], the parameter vector θ is learned using back-propagation to optimise the classification loss (*e.g.*, cross-entropy).

Singular Value Decomposition (SVD) Singular Value Decomposition (SVD) is a factorisation of a matrix that generalises the eigendecomposition of square matrices to any rectangular matrix. Given $\mathbf{W} \in \mathbb{R}^{s \times r}$ with rank $k \leq \min(s, r)$, we denote $\{\sigma_i\}_{i=1}^k$, $\{\mathbf{u}_i\}_{i=1}^k$, and $\{\mathbf{v}_i\}_{i=1}^k$ as its singular values, left singular vectors, and right singular vectors, respectively. Throughout this thesis, a set of singular values is assumed to be sorted $\sigma_1 \geq \cdots \geq \sigma_k$. $\sigma_i(\mathbf{W})$ denotes the i-th singular value of the matrix $\mathbf{W}$. Using singular values, the matrix 2-norm $\|\mathbf{W}\|_2$ and the Frobenius norm $\|\mathbf{W}\|_\mathrm{F}$ can be computed as σ_1 and $\sqrt{\sum_i \sigma_i^2}$, respectively. We discuss this further, along with other linear algebra basics, in Appendix A.

4.2.1 Stable Rank

Below we provide the formal definition and some properties of stable rank.

Definition 18 (Stable Rank)**.** *The Stable Rank [224] of an arbitrary matrix* $\mathbf{W}$ *is defined as* $\mathrm{srank}(\mathbf{W}) = \frac{\|\mathbf{W}\|_\mathrm{F}^2}{\|\mathbf{W}\|_2^2} = \frac{\sum_{i=1}^k \sigma_i^2(\mathbf{W})}{\sigma_1^2(\mathbf{W})}$*, where* k *is the rank of the matrix. Stable rank is*

1. *a soft version of the rank operator and, unlike rank, is less sensitive to small perturbations.*
2. *almost always differentiable as both Frobenius and Spectral norms are almost always differentiable.*
3. *upper-bounded by the rank of the matrix:* $\mathrm{srank}(\mathbf{W}) = \frac{\sum_{i=1}^k \sigma_i^2(\mathbf{W})}{\sigma_1^2(\mathbf{W})} \leq \frac{\sum_{i=1}^k \sigma_1^2(\mathbf{W})}{\sigma_1^2(\mathbf{W})} = k$*.*
4. *invariant to scaling, i.e. for any* $\eta \in \mathbb{R} \setminus \{0\}$ *we have that* $\mathrm{srank}(\mathbf{W}) = \mathrm{srank}(\frac{\mathbf{W}}{\eta})$*.*

[2] y_j is the j-th element of vector $\mathbf{y}$. Only one class is assigned as the ground-truth label to each $\mathbf{x}$.

4.2.2 Lipschitz Constant

Lipschitzness of a function is an indication of the smoothness of the function. The Lipschitz constant of a function is a quantification of the sensitivity of the output of the function with respect to the change in the input. Thus, functions with small lipschitz constants are smoother than functions with large lipschitz constants. This section first describes the concepts of global lipschitz, local lipschitz, and empirical lipschitz constants and then discusses why the product of spectral norms is an upper bound on the local lipschitzness of neural networks.

Global and Local Lipschitzness A function $f : \mathbb{R}^d \mapsto \mathbb{R}^k$ is *globally L-Lipschitz continuous* if there exists $L \in \mathbb{R}_+$ such that $\|\mathrm{f}(\mathbf{x}_\mathrm{i}) - \mathrm{f}(\mathbf{x}_\mathrm{j})\|_q \leq L\|\mathbf{x}_\mathrm{i} - \mathbf{x}_\mathrm{j}\|_p$ for all $(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}^d$, where $\|\cdot\|_p$ and $\|\cdot\|_q$ represents the ℓ_p and ℓ_q norms in the input and the output metric spaces, respectively. The global Lipschitz constant L_g is:

$$L_g = \max_{\substack{\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d \\ \mathbf{x}_i \neq \mathbf{x}_j}} \frac{\|\mathrm{f}(\mathbf{x}_\mathrm{i}) - \mathrm{f}(\mathbf{x}_\mathrm{j})\|_q}{\|\mathbf{x}_\mathrm{i} - \mathbf{x}_\mathrm{j}\|_p}. \tag{4.1}$$

The above definition of the Lipschitz constant accounts for all pairs of inputs in the domain $\mathbb{R}^d \times \mathbb{R}^d$. It is thus said to be the *global lipschitz constant*. One can define the local Lipschitz constant based on the sensitivity of f in the vicinity of a given point $\mathbf{x}$.

For a given $\mathbf{x}$ and for an arbitrarily small $\delta > 0$, the local Lipschitz constant is computed in an open ball of radius δ centred at $\mathbf{x}$. Let $\mathbf{h} \in \mathbb{R}^d$ with $\|\mathbf{h}\|_p < \delta$, then, similar to L_g, the *local lipschitz constant* of f at $\mathbf{x}$, $L_l(\mathbf{x})$, is greater than or equal to

$$\sup_{\mathbf{h} \neq 0, \|\mathbf{h}\|_p < \delta} \frac{\|\mathrm{f}(\mathbf{x} + \mathbf{h}) - \mathrm{f}(\mathbf{x})\|_q}{\|\mathbf{h}\|_p} \tag{4.2}$$

Assuming f to be Fréchet differentiable, as $\mathbf{h}$ tends to 0, we can use the first-order approximation on f: $f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) \approx J_f(\mathbf{x})\mathbf{h}$, where $J_f(\mathbf{x}) = \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}|_{\mathbf{x}} \in \mathbb{R}^{k \times d}$ is the jacobian of f at $\mathbf{x}$. Then the local lipschitz constant of f at $\mathbf{z}$ is the matrix (operator) norm of the Jacobian $J_f(\mathbf{x})$.

$$L_l(\mathbf{x}) \overset{(a)}{=} \lim_{\delta \to 0} \sup_{\substack{\mathbf{h} \neq 0 \\ \|\mathbf{h}\|_p < \delta}} \frac{\|\mathrm{J_f}(\mathbf{x})\mathbf{h}\|_q}{\|\mathbf{h}\|_p} \overset{(b)}{=} \sup_{\substack{\mathbf{h} \neq 0 \\ \mathbf{h} \in \mathbb{R}^m}} \frac{\|\mathrm{J_f}(\mathbf{x})\mathbf{h}\|_q}{\|\mathbf{h}\|_p} = \|\mathrm{J_f}(\mathbf{x})\|_{p,q}^{\mathrm{op}}. \tag{4.3}$$

Here, (a) is by definition of local lipschitzness and (b) is due to the property of norms that for any non-negative scalar c, $\|c\mathbf{x}\| = c\|\mathbf{x}\|$. Note that $\|\mathrm{J_f}(\mathbf{x})\|_{p,q}^{\mathrm{op}}$ is the p, q matrix operator norm, defined in Equation (A.2), of the Jacobian matrix and is different from the entry-wise p, q norm, which is for example used in Equation (3.6). A function is said to be *locally Lipschitz* with *local Lipschitz constant* L_l if, for all $\mathbf{x} \in \mathbb{R}^d$, the function is L_l *locally-Lipschitz* at $\mathbf{x}$. Thus,

$$L_l = \sup_{\mathbf{x} \in \mathbb{R}^d} L_l(\mathbf{x}) \tag{4.4}$$

Notice that the Lipschitz constant (global or local) greatly depends on the chosen norms. When $p = q = 2$, the upper bound on the local Lipschitz constant at $\mathbf{x}$ boils down to the 2-matrix norm (maximum singular value) of the Jacobian $J_f(\mathbf{x})$ (see last equality of Equation (4.3)).

The local Lipschitz upper-bound for Neural Networks Here we show that the local lipschitz constant for neural networks can be upper bounded by the product of spectral norms of individual weight matrices. Interestingly this upper-bound is data-agnostic.

Lemma 1 (Local Lipschitz upper bound for NNs)**.** *For a neural network f belonging to the class of multi-layered perceptrons, as defined in Definition 1, the local lipschitzness at a point $\mathbf{x} \in \mathbb{R}^d$ can be upper-bounded as*

$$L_l(\mathbf{x}) =\leq \|\mathbf{W}_1\|_{p,q}^{\mathrm{op}} \cdots \|\mathbf{W}_{\mathrm{L}}\|_{p,q}^{\mathrm{op}} \tag{4.5}$$

Further, as the upper-bound is data-agnostic, the local lipschitz constant is also equal to this upper-bound i.e.

$$L_l = L_l(\mathbf{x})$$

Proof in Appendix C.1.

Next we discuss more optimistic (or empirical) estimates of L_l and L_g, its link with generalisation and then in Section 4.4, we show empirically the effect of SRN on empirical lipschitzness and generalisation.

Empirical Lipschitz constants It is clear from Lemma 1 that the Lipschitz constant upper bound ($\prod_i^L \|\mathbf{W}_\mathrm{i}\|_2$), along with being scale-dependent, is also *data-independent* and hence, provides a pessimistic estimate of the behaviour of a model on a particular task or dataset. We call this pessimistic as the behaviour of the model on the entire input domain is not always relevant especially when data lies in a more restricted portion of the domain. Considering this, a relatively optimistic estimate of the model's behaviour would be an *empirical* estimate of the Lipschitz constant (L_e) on a task-specific dataset. The global and local L_e are simply the equivalent of the global and local lipschitz constant defined in Equation (4.1) and Equation (4.4) respectively with the maximisation done on just the support of the data distribution as opposed to the whole of $\mathbb{R}^d$. Note that local L_e is just the norm of the Jacobian at a given point. For completeness, we provide the relationship between the global and the local L_e in Lemma 2.

Lemma 2 (Relating Empirical and Global Lipschitzness)**.** *Let $f : \mathbb{R}^d \mapsto \mathbb{R}$ be a Fréchet differentiable function, $\mathcal{D}$ the dataset, and Conv $(\mathbf{x}_i, \mathbf{x}_j)$ denotes the convex combination of a pair of samples $\mathbf{x}_i$ and $\mathbf{x}_j$, then $\forall p, q \in [1, \infty]$ such that $\frac{1}{p} + \frac{1}{q} = 1$*

$$\max_{\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}} \frac{|f(\mathbf{x}_i) - f(\mathbf{x}_j)|}{\|\mathbf{x}_\mathrm{i} - \mathbf{x}_\mathrm{j}\|_p} \leq \max_{\substack{\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D} \\ \mathbf{x} \in Conv\,(\mathbf{x}_i, \mathbf{x}_j)}} \|\mathrm{J_f}(\mathbf{x})\|_q$$

Proof in Appendix C.1.

As discussed before, the local lipschitz constant upper bound in Equation (4.5) is independent of $\mathbf{x}$. This is one of the main reasons why we consider the empirical Lipschitz to better reflect the true behaviour of the function as the NN is never exposed to the entire domain $\mathbb{R}^d$ but only a small subset dependent on the data distribution. The other reason why this upper bound is a bad estimate is that the inequality in Eq (4.5) is tight only when the partial derivatives are aligned, implying,

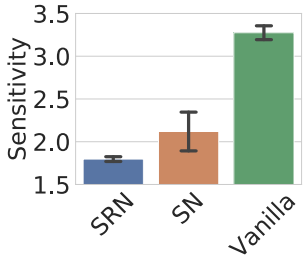


Figure 4.1: Noise Sensitivity (lower the better). Test accuracy: SRN (73.1%), SN (71.5%), and Vanilla (72.4%).

$\left\| \frac{\partial \mathbf{z}_\ell}{\partial \mathbf{z}_{\ell-1}} \frac{\partial \mathbf{z}_{\ell+1}}{\partial \mathbf{z}_\ell} \right\|_2 = \left\| \frac{\partial \mathbf{z}_\ell}{\partial \mathbf{z}_{\ell-1}} \right\|_2 \left\| \frac{\partial \mathbf{z}_{\ell+1}}{\partial \mathbf{z}_\ell} \right\|_2 \quad \forall l-2 \leq \ell \leq l$. This problem has been referred to as the problem of mis-alignment and is similar to quantities like layer cushion in Arora et al. [16].

4.3 Stable Rank Normalization

This section describes a technique, we call Stable Rank Normalization (SRN), to control the stable rank of linear operators. A big challenge in stable rank normalization comes from the fact that stable rank is scale-invariant (refer to definition 18), thus, any normalization scheme that modifies $\mathbf{W} = \sum_i \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ to $\widehat{\mathbf{W}} = \sum_i \frac{\sigma_i}{\eta} \mathbf{u}_i \mathbf{v}_i^\top$ (for any $\eta > 0$ will not affect on the stable rank. Examples of such schemes are SN [182] where $\eta = \sigma_1$, and Frobenius normalization where $\eta = \|\mathbf{W}\|_{\mathrm{F}}$. But first, we look at some conceptual motivation for controlling stable rank.

4.3.1 Impact of stable rank on noise-sensitivity

Stable rank controls the noise-sensitivity As shown by Arora et al. [16], one of the critical properties of generalisable NNs is low noise sensitivity— the ability of a network to preferentially carry over the true signal in the data. For a given noise distribution $\mathcal{N}$, it can be quantified as

$$\Phi_{f_\theta,\mathcal{N}} = \max_{\mathbf{x}\in\mathcal{D}} \Phi_{f_\theta,\mathcal{N}}(\mathbf{x}), \quad \textit{where} \quad \Phi_{f_\theta,\mathcal{N}}(\mathbf{x}) := \mathbb{E}_{\eta\sim\mathcal{N}} \left[\frac{\|\mathrm{f}_\theta(\mathbf{x}+\eta\|\mathbf{x}\|) - \mathrm{f}_\theta(\mathbf{x})\|^2}{\|\mathrm{f}_\theta(\mathbf{x})\|^2} \right] \tag{4.6}$$

For a linear mapping with parameters $\mathbf{W}$ and the noise distribution being normal-$\mathcal{N}(0, \mathbf{I})$, it can be shown that $\Phi_{f_\mathbf{W},\mathcal{N}} \geq \mathrm{srank}(\mathbf{W})$ (see Proposition 3.1 in Arora et al. [16]). Thus, decreasing the stable rank decreases this lower bound on noise sensitiv-

ity. In Figure 4.1, we show $\Phi_{f_\theta,\mathcal{N}}$ of a ResNet110 trained on CIFAR100. Note that although the Lipschitz upper bound (Lemma 1) of SRN and SN are the same, SRN (algorithmic details in Section 4.3) is much less sensitive to noise than SN. This can be possibly explained by SRN's impact on the empirical lipschitz constant, which we discuss below.

Stable rank impacts empirical lipschitz constant Empirically, Novak et al. [199] provided results showing how local empirical lipschitz L_e (in the vicinity of train data) is correlated with the generalisation error of neural networks. This observation is further supported by the theoretical works of Wei and Ma [269], Nagarajan and Kolter [192], and Arora et al. [16] whereby variants of L_e are used to derive generalisation bounds for neural networks. Thus, a tool that favours low L_e is likely to provide better generalisation behaviour in practice. A low local empirical lipschitz also decreases the noise-sensitivity of neural networks (c.f. Equation (4.6)).

To this end, we first consider a simple two layer linear neural network and show that low rank transformations favour low L_e. A linear neural network is a neural network with linear activation functions. Since direct minimisation of rank for NNs is non-trivial, the expectation is that learning weight matrices with low stable rank (softer version of rank) might induce similar behaviour. We experimentally validate this hypothesis by showing that, as we decrease the stable rank, the empirical Lipschitz decreases. This shows SRN indeed prefers learning transformations with low empirical Lipschitz constant.

Let $f(\mathbf{x}) = \mathbf{W}_2\mathbf{W}_1\mathbf{x}$ be a two-layer linear neural network with weights $\mathbf{W}_1$ and $\mathbf{W}_2$. The Jacobian, in this case, is independent of $\mathbf{x}$. Thus, the local Lipschitz constant is the same for all $\mathbf{x} \in \mathbb{R}^d$, implying, local $L_e = L_l(\mathbf{x}) = L_l = \|\mathbf{W}_2\mathbf{W}_1\| \leq \|\mathbf{W}_2\|\|\mathbf{W}_1\|$. Note, in the case of the 2-matrix norm, reducing the rank of $\mathbf{W_1}$ and $\mathbf{W}_2$ does not affect the upper bound $\|\mathbf{W}_1\|_2\|\mathbf{W}_2\|_2$. However, we discuss below that rank reduction influences the global L_e.

Let $\mathbf{x}_i$ and $\mathbf{x}_j$ be a random pair from the dataset $\mathcal{D}$ and $\Delta\mathbf{x} \neq \mathbf{0}$ be the difference $\mathbf{x}_i - \mathbf{x}_j$. Then, the global L_e is $\max_{\{\mathbf{x}_i,\mathbf{x}_j\}\in\mathcal{D}} \frac{\|\mathbf{W}_2\mathbf{W}_1\Delta\mathbf{x}\|}{\|\Delta\mathbf{x}\|}$. Let k_1 and k_2 be the ranks, and $\sigma_1 \geq \cdots \geq \sigma_{k_1}$ and $\lambda_1 \geq \cdots \geq \lambda_{k_2}$ be the singular values of the matrices $\mathbf{W}_1$ and

$\mathbf{W}_2$, respectively. Let $P_i = \mathbf{u}_i \bar{\mathbf{u}}_i^\top$ be the orthogonal projection matrix corresponding to $\mathbf{u}_i$ and $\bar{\mathbf{u}}_i$, the left and the right singular vectors of $\mathbf{W}_1$. Similarly, we define Q_i for $\mathbf{W}_2$ corresponding to $\mathbf{v}_i$ and $\bar{\mathbf{v}}_i$. Then, the product of the two matrices can be decomposed into $\mathbf{W}_2\mathbf{W}_1 = \sum_{i=1}^{k_2}\sum_{j=1}^{k_1} \lambda_i \sigma_j Q_i P_j$. The largest singular value of the product is equal to its maximum possible value $\lambda_1\sigma_1$ if and only if $\Delta\mathbf{x} = \bar{\mathbf{u}}_1\|\Delta\mathbf{x}\|$ and $\mathbf{u}_1 = \bar{\mathbf{v}}_1$, i.e. a perfect alignment of the top most singular vectors of $\mathbf{W}_1$ and $\mathbf{W}_2$ occurs, which is highly unlikely. In practice, not just the maximum singular values, unlike the case with the Lipschitz upper-bound in Lemma 1, rather the combination of the projection matrices and the singular values play a crucial role in providing an estimate of global L_e. Thus, zeroing out certain singular values, which is equivalent to minimizing the rank (or stable rank), reduces L_e. For example, assigning $\sigma_j = 0$, which in effect reduces the rank of $\mathbf{W}_1$ by one, nullifies the contribution of all projections of $\mathbf{W}_2$ on P_j. Thus, all k_2 projections $\sigma_j(\sum_{i=1}^{k_2} \lambda_i Q_i)P_j$ that would have propagated the input via P_j are blocked. This influences $\|\mathbf{W}_2\mathbf{W}_2\Delta\mathbf{x}\|$ and hence, the global L_e. In a more general setting, if k_i is the rank of the i^{th} linear layer, then, each singular value of the j-th layer can influence a maximum of $\prod_{i=1}^{j-1} k_i \prod_{i=j+1}^{l} k_i$ many paths through which an input can propagate. Thus, transformations with low rank (stable) reduce the global L_e. Similar arguments can be drawn for local L_e in the case of NN with non-linearity.

4.3.2 Solution to the Stable Rank Normalization problem

In this section, we will define and solve the stable rank normalization problem.

The SRN problem statement Given a matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ with rank p and a spectral partitioning index k $(0 \leq k < p)$, we formulate the SRN problem as

$$\underset{\widehat{\mathbf{W}}_k \in \mathbb{R}^{m\times n}}{\mathbf{argmin}} \left\| \mathbf{W} - \widehat{\mathbf{W}}_{\mathrm{k}} \right\|_{\mathrm{F}}^2 \quad s.t. \quad \underbrace{\mathrm{srank}(\widehat{\mathbf{W}}_k) = r}_{\text{stable rank constraint}}, \ \underbrace{\lambda_i = \sigma_i, \forall i \in \{1, \cdots, k\}}_{\text{spectrum preservation constraints}} . \quad (4.7)$$

where, $1 \leq r < \mathrm{srank}(\mathbf{W})$ is the desired stable rank, λ_is and σ_is are the singular values of $\widehat{\mathbf{W}}_k$ and $\mathbf{W}$, respectively. The partitioning index k is the *singular value (or the spectrum) preservation constraint.* It gives us the flexibility to obtain $\widehat{\mathbf{W}}_k$ such that its top k singular values are the same as that of the original matrix. Note

that the problem statement is more general in the sense that putting $k = 0$ removes the spectrum preservation constraint entirely.

The solution to SRN The optimal unique solution to the above problem is provided in Theorem 1 and proved in Appendix C.2. At $k = 0$, the problem (4.7) is non-convex, otherwise convex.

Theorem 1 (Solution to the Stable Rank Problem)**.** *Given a real matrix* $\mathbf{W} \in \mathbb{R}^{m\times n}$ *with rank* p*, a target spectrum (or singular value) preservation index* k $(0 \leq k < p)$*, and a target stable rank* r $(1 \leq r < \text{srank}(\mathbf{W}))$*, the optimal solution* $\widehat{\mathbf{W}}_k$ *to the optimisation problem in Equation* (4.7) *is* $\widehat{\mathbf{W}}_k = \gamma_1\mathbf{S}_1 + \gamma_2\mathbf{S}_2$*, where* $\mathbf{S}_1 = \sum_{i=1}^{\max(1,k)} \sigma_i\mathbf{u}_i\mathbf{v}_i^\top$*,* $\mathbf{S}_2 = \mathbf{W} - \mathbf{S}_1$*.* $\{\sigma_i\}_{i=1}^k$*,* $\{\mathbf{u}_i\}_{i=1}^k$ *and* $\{\mathbf{v}_i\}_{i=1}^k$ *are the top* k *singular values and vectors of* $\mathbf{W}$*, and, depending on* k*,* γ_1 *and* γ_2 *are defined below. For simplicity, we first define* $\gamma = \frac{\sqrt{r\sigma_1^2 - \|\mathbf{S}_1\|_\text{F}^2}}{\|\mathbf{S}_2\|_\text{F}}$*, then*

a) If $k = 0$ *(no spectrum preservation), the problem is non-convex, the optimal solution to which is obtained for* $\gamma_2 = \dfrac{\gamma + r - 1}{r}$ *and* $\gamma_1 = \dfrac{\gamma_2}{\gamma}$*, when* $r > 1$*. If* $r = 1$*, then* $\gamma_2 = 0$ *and* $\gamma_1 = 1$*. In particular as* $\|\mathbf{S}_1\|_\text{F}^2 = \sigma_1^2$*, we get* $\gamma = \frac{\sqrt{r-1}\sigma_1}{\|\mathbf{S}_2\|_\text{F}}$*.*

b) If $k \geq 1$*, the problem is convex. If* $r \geq \frac{\|\mathbf{S}_1\|_\text{F}^2}{\sigma_1^2}$ *the optimal solution is obtained for* $\gamma_1 = 1$*, and* $\gamma_2 = \gamma$ *and if not, the problem is not feasible.*

c) Also, $\left\|\widehat{\mathbf{W}}_\text{k} - \mathbf{W}\right\|_\text{F}$ *monotonically increases with* k *for* $k \geq 1$*.*

Proof in Appendix C.2

Intuitively, Theorem 1 partitions the given matrix into two parts, depending on k, and then scales them differently to obtain the optimal solution. The value of the partitioning index k is a design choice. If there is no particular preference for k, then $k = 0$ provides the most optimal solution in terms of closeness to the original matrix. We provide a simple example to demonstrate this. Given $\mathbf{W} = \mathbf{I}_3$ (rank $=$ srank$(\mathbf{W}) = 3$), the objective is to project it to a new matrix with stable rank of 2. Consider the following three solutions to the problem. $\widehat{\mathbf{W}}_1$ is obtained using the standard rank minimisation (Eckart-Young-Mirsky [75]) while $\widehat{\mathbf{W}}_2$ and $\widehat{\mathbf{W}}_3$ are the

solutions of Theorem 1 with $k = 1$ and $k = 0$, respectively.

$$\widehat{\mathbf{W}}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \widehat{\mathbf{W}}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}, \widehat{\mathbf{W}}_3 = \begin{bmatrix} \frac{\sqrt{2}+1}{2} & 0 & 0 \\ 0 & \frac{\sqrt{2}+1}{2\sqrt{2}} & 0 \\ 0 & 0 & \frac{\sqrt{2}+1}{2\sqrt{2}} \end{bmatrix} \tag{4.8}$$

It is easy to verify that the stable rank of all the above solutions is 2. However, the Frobenius distance (lower the better) of these solutions from the original matrix follows the order $\left\|\mathbf{W} - \widehat{\mathbf{W}}_1\right\|_{\mathrm{F}} > \left\|\mathbf{W} - \widehat{\mathbf{W}}_2\right\|_{\mathrm{F}} > \left\|\mathbf{W} - \widehat{\mathbf{W}}_3\right\|_{\mathrm{F}}$. Thus, Theorem 1 with $k = 0$ provides the closest solution followed by $k = 1$ and then the hard rank solution $\widehat{\mathbf{W}}_1$. As evident from the example, the solution to SRN, instead of completely removing a particular singular value like $\widehat{\mathbf{W}_1}$, scales them (depending on k) such that the new matrix has the desired stable rank. Note that for $\widehat{\mathbf{W}}_1$ and $\widehat{\mathbf{W}}_2$ (true for any $k \geq 1$), the spectral norm of the original and the normalized matrices are the same, implying, $\gamma_1 = 1$. However, for $k = 0$, the spectral norm of the optimal solution is greater than that of the original matrix. It is easy to verify from Theorem 1 that as k increases, γ_2 decreases. Thus, the amount of scaling required for the second partition $\mathbf{S}_2$ is more aggressive. In all situations, the following inequality holds: $\gamma_2 \leq 1 \leq \gamma_1$.

Optimal Spectral Normalization The widely used spectral normalization [182] where the given matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ is divided by the maximum singular value is an approximation to the optimal solution of the spectral normalization problem defined as

$$\begin{aligned} &\underset{\widehat{\mathbf{W}}}{\mathbf{argmin}} \left\|\mathbf{W} - \widehat{\mathbf{W}}\right\|_{\mathrm{F}}^2 \\ &s.t. \quad \sigma(\widehat{\mathbf{W}}) \leq s, \end{aligned} \tag{4.9}$$

where $\sigma(\widehat{\mathbf{W}})$ denotes the maximum singular value and $s > 0$ is a hyperparameter. The optimal solution to this problem is shown in Algorithm 3. Here, we provide the proof of optimality of Algorithm 3 for the sake of completeness.

Let $\text{SVD}\,(\mathbf{W}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and let us assume that $\mathbf{Z} = \mathbf{S}\Lambda\mathbf{T}^\top$ is a solution to the

Algorithm 3 Spectral Normalization

input $\mathbf{W} \in \mathbb{R}^{m \times n}$, s

$\mathbf{W}_1 \leftarrow \mathbf{0}$, $p \leftarrow \min(m, n)$
for $k \in \{1, \cdots, p\}$ **do**
  $\{\mathbf{u}_k, \mathbf{v}_k, \sigma_k\} \leftarrow SVD(\mathbf{W}, k)$ {perform power method to get k-th singular value}
  if $\sigma_k \geq s$ **then**
    $\mathbf{W}_1 \leftarrow \mathbf{W}_1 + s\ \mathbf{u}_k \mathbf{v}_k^\top$
    $\mathbf{W} \leftarrow \mathbf{W} - \sigma_k\ \mathbf{u}_k \mathbf{v}_k^\top$
  else
    break {exit for loop}
  end if
end for

output $\mathbf{W} \leftarrow \mathbf{W}_1 + \mathbf{W}$

problem 4.9. Trivially, $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top$ also satisfies $\sigma(\mathbf{X}) \leq s$. Now, $\|\mathbf{W} - \mathbf{X}\|_{\mathrm{F}}^2 = \left\|\mathbf{U}(\Sigma - \Lambda)\mathbf{V}^\top\right\|_{\mathrm{F}}^2 = \|(\Sigma - \Lambda)\|_{\mathrm{F}}^2 \leq \|\mathbf{W} - \mathbf{Z}\|_{\mathrm{F}}^2$, where the last inequality directly comes from Lemma A. Thus the singular vectors of the optimal solution must be the same as that of $\mathbf{W}$. This boils down to solving the following problem

$$\underset{\mathbf{\Lambda} \in \mathbb{R}_+^{\min(m,n)}}{\mathbf{argmin}} \ \|\mathbf{\Lambda} - \mathbf{\Sigma}\|_{\mathrm{F}}^2 \ s.t.\ \mathbf{\Lambda}[i] \leq s \ \ \forall i \in \{0, \min(m, n)\}. \tag{4.10}$$

Here, without loss of generality, we abuse notations by considering $\mathbf{\Lambda}$ and $\mathbf{\Sigma}$ to represent the diagonal vectors of the original diagonal matrices $\mathbf{\Lambda}$ and $\mathbf{\Sigma}$, and $\mathbf{\Lambda}[\mathbf{i}]$ as its i-th index. It is trivial to see that the optimal solution with minimum Frobenius norm is achieved when

$$\mathbf{\Lambda}[i] = \begin{cases} \mathbf{\Sigma}[\mathbf{i}], & \text{if } \ \mathbf{\Sigma}[i] \leq s \\ s, & \text{otherwise.} \end{cases}$$

This is exactly what Algorithm 3 implements.

4.3.3 Algorithm for Stable Rank Normalization (SRN)

We provide a general procedure in Algorithm 4 to solve the stable rank normalization problem for $k \geq 1$ (the solution for $k = 0$ is straightforward from Theorem 1). Claim 1 provides the properties of the algorithm. The algorithm is constructed so that the prior knowledge of the rank of the matrix is not necessary.

Claim 1. *Given a matrix* $\mathbf{W}$*, the desired stable rank* r*, and the partitioning index* $k \geq 1$*, Algorithm 4 requires computing the top* l $(l \leq k)$ *singular values and vectors of* $\mathbf{W}$*. It returns* $\widehat{\mathbf{W}}_l$ *and the scalar* l *such that* $\text{srank}(\widehat{\mathbf{W}}_l) = r$*, and the top* l *singular values of* $\mathbf{W}$ *and* $\widehat{\mathbf{W}}_l$ *are the same. If* $l = k$*, then the solution provided is the optimal solution to the problem* (4.7) *with all the constraints satisfied, otherwise, it returns the largest* l *up to which the spectrum is preserved.*

Algorithm 4 Stable Rank Normalization

input $\mathbf{W} \in \mathbb{R}^{m \times n}$, r, $k \geq 1$
$\mathbf{S}_1 \leftarrow \mathbf{0}$, $\beta \leftarrow \|\mathbf{W}\|_{\text{F}}^2$, $\eta \leftarrow 0$, $l \leftarrow 0$
for $i \in \{1, \cdots, k\}$ **do**
  $\{\mathbf{u}_i, \mathbf{v}_i, \sigma_i\} \leftarrow SVD(\mathbf{W}, i)$ {Power method to get i-th singular value}
  if $r \geq \left(\sigma_i^2 + \eta\right)/\sigma_1^2$ **then**
    $\mathbf{S}_1 \leftarrow \mathbf{S}_1 + \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$
    $\eta \leftarrow \eta + \sigma_i^2, \beta \leftarrow \beta - \sigma_i^2$
    $l \leftarrow l + 1$
  else
    break
  end if
end for
$\eta \leftarrow r\sigma_1^2 - \eta$
output $\widehat{\mathbf{W}}_l \leftarrow \mathbf{S}_1 + \sqrt{\frac{\eta}{\beta}}(\mathbf{W} - \mathbf{S}_1)$, l

Algorithm 5 SRN for a Linear Layer in NN

input $\mathbf{W} \in \mathbb{R}^{m \times n}$, r, learning rate α, mini-batch dataset $\mathcal{D}$
Initialize $\mathbf{u} \in \mathbb{R}^m$ with a random vector.
$\mathbf{v} \leftarrow \frac{\mathbf{W}^\top \mathbf{u}}{\|\mathbf{W}^\top \mathbf{u}\|}$, $\mathbf{u} \leftarrow \frac{\mathbf{W}^\top \mathbf{v}}{\|\mathbf{W}^\top \mathbf{v}\|}$ {Perform power iteration}
$\sigma(\mathbf{W}) = \mathbf{u}^\top \mathbf{W} \mathbf{v}$
$\mathbf{W}_f = \mathbf{W}/\sigma(\mathbf{W})$ {Spectral Normalization}
$\widehat{\mathbf{W}} = \mathbf{W}_f - \mathbf{u}\mathbf{v}^\top$
if $\left\|\widehat{\mathbf{W}}\right\|_{\text{F}} \leq \sqrt{r-1}$ **then**

output $\mathbf{W}_f$
end if
$\mathbf{W}_f = \mathbf{u}\mathbf{v}^\top + \widehat{\mathbf{W}} \frac{\sqrt{r-1}}{\left\|\widehat{\mathbf{W}}\right\|_{\text{F}}}$ {Stable Rank Normalization}

output $\mathbf{W} \leftarrow \mathbf{W} - \alpha \nabla_{\mathbf{W}} L(\mathbf{W}_f, \mathcal{D})$

Combining Stable Rank and Spectral Normalization for NNs Following the arguments provided in Chapter 1 and Section 4.3.1, for better generalisability, we propose to normalize *both* the stable rank and the spectral norm of each linear layer of a NN simultaneously. To do so, we first perform approximate SN [182], and then perform optimal SRN (using Algorithm 4). We use $k = 1$ to ensure that the first singular value (which is now normalized) is preserved. Algorithm 5 provides a simplified procedure for the same for a given linear layer of a NN. Note that the computational cost of this algorithm is *exactly the same as that of SN*, which is to compute the top singular value using the power iteration method.

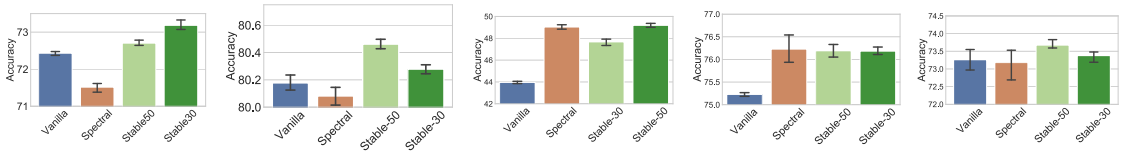
Implementation details of Stable Rank Normalization The SRN algorithm (Algorithm 5) is applied on neural networks in exactly the same way as Spectral Nor-

malization (SN) is applied in Miyato et al. [182]. In particular, every iteration of gradient descent (or its variant) conducts one forward pass followed by a backward pass to compute gradients, and then the parameters are updated according to the update step of the optimisation algorithm. Finally, each weight matrix is projected onto the space of matrices with constrained stable rank by applying Algorithm 5 on each weight matrix.

Remark 3. *While applying Algorithm 5 on fully connected layers is straightforward, for convolution layers, we apply SRN on a matrix corresponding to the linear transformation of the convolution operation, similar to Miyato et al. [182]. For a convolution layer with weight* $W \in \mathbb{R}^{c_i \times c_o \times h \times w}$ *where* c_i *and* c_o *are the number of input and output filters respectively, and* $h \times w$ *is the dimension of individual filters, we flatten the layer into a* $(c_i, c_o hw)$*-dimensional matrix and apply Algorithm 5 on this matrix. While this is not an exact representation of the linear transformation of a convolutional layer, this heuristic helps in the computational speed of the operation. We also note that this framework is not accurate for residual networks. We apply SRN on the learnable layers inside the residual block, however the actual transformation of a residual block also includes the identity map applied by the skip connection. It is not straightforward whether this can be represented by a linear transformation as there are also non-linear activations inside the residual block, which the skip connection bypasses. However, despite these shortcomings, our method works well in practice as the next section shows.*

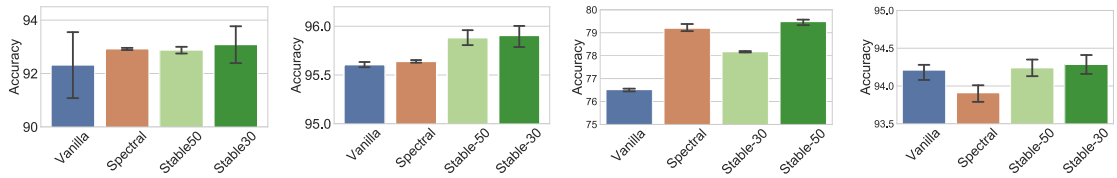
4.4 Experiments on a discriminative setup

Dataset and architectures For classification, we perform experiments on ResNet-110 [110], WideResNet-28-10 [289], DenseNet-100 [119], VGG-19 [239], and AlexNet [145] using the CIFAR100 and CIFAR10 [145] datasets. We train them using standard training recipes with SGD, using a learning rate of 0.1 (except AlexNet where we use a learning rate of 0.01), and a momentum of 0.9 with a batch size of 128 (further details in Appendix B). In addition to training for a fixed number of epochs, we also present results where the training accuracy (as opposed to the number of iterations)



(a) ResNet110 (b) WideResNet28 (c) Alexnet (d) Densenet-100 (e) VGG-19

Figure 4.2: Test accuracies on CIFAR100 for clean data using the number of epochs as stopping criterion. Higher is better.



(a) ResNet110 (b) WideResNet28 (c) Alexnet (d) Densenet-100

Figure 4.3: Test accuracies on CIFAR10 for clean data using the number of epochs as stopping criterion. Higher is better.

is used as a stopping criterion to show that our regulariser performs well with a range of stopping criteria.

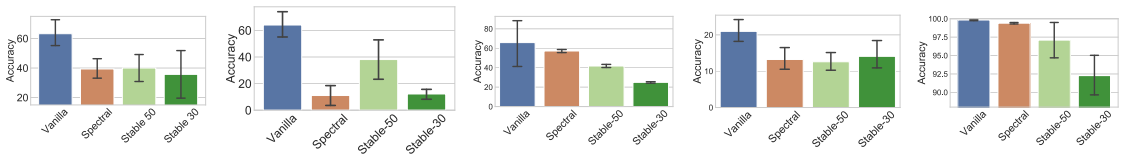
For GAN experiments, we use CIFAR100, CIFAR10, and CelebA [168] datasets. We show results on both, conditional and unconditional GANs. Please refer to Appendix B for further details about the training setup.

Choosing stable rank Given a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, the desired stable rank r is controlled using a single hyperparameter c as $r = c \ \min(m, n)$, where $c \in (0, 1]$. For simplicity, we use the same c for all the linear layers. Note that if $c = 1$, or for a given c, if $\text{srank}(\mathbf{W}) \leq r$, then SRN boils down to SN. For classification, we choose $c = \{0.3, 0.5\}$, and compare SRN against standard training (Vanilla) and training with SN. For GAN experiments, we choose $c = \{0.1, 0.3, 0.5, 0.7, 0.9\}$, and compare SRN-GAN against SN-GAN [182], WGAN-GP [97], and orthonormal regularisation GAN (Ortho-GAN) [38].

4.4.1 Classification experiments

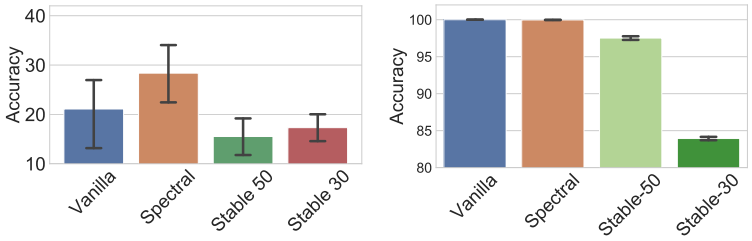
We perform each experiment 5 times using a new random seed each time and report the mean and the 75% confidence interval for the test error. We plot the test accuracy on CIFAR100 in Figure 4.2 and CIFAR10 in Figure 4.3.

These experiments show that the test accuracy of SRN on a wide variety NNs is



(a) ResNet110 (b) WideResNet28 (c) Alexnet (d) Densenet-100 (e) VGG-19

Figure 4.4: Train accuracies on CIFAR100 for shattering experiment. Lower indicates less memorisation, thus, better.



(a) ResNet110 (b) Alexnet

Figure 4.5: Train accuracies on CIFAR10 for shattering experiment. Lower indicates less memorisation, thus, better.

always higher than the Vanilla and SN (except for SRN-50 on Alexnet where SRN and SN are almost equal). However, SN performs slightly worse than Vanilla for WideResNet-28 and ResNet110. The fact that SRN also involves SN, combined with the above observation, indicates that even though SN reduced the learning capability of these networks, normalizing stable rank must have improved it significantly in order for SRN to outperform Vanilla. For example, in the case of ResNet110, SN is 71.5% accurate whereas SRN provides an accuracy of 73.2%. In addition to this, we also note that even though SN is used extensively for the training of GANs, it is not a popular choice when it comes to training standard NNs for classification. We suspect that this is because of the decrease in performance, we observe here. Hence, as our experiments indicate that SRN overcomes this, SRN could be a potentially important regulariser in the classification setting.

4.4.2 Shattering experiments

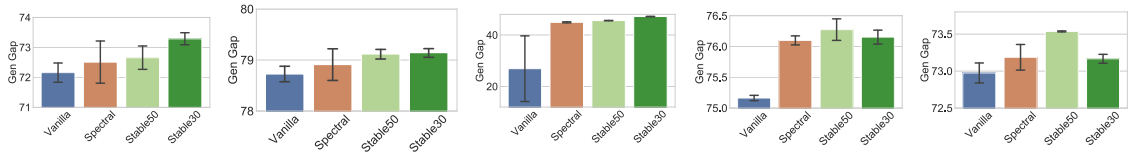
Our previous set of experiments established that SRN provides improved classification accuracies on various NNs. Here we study the generalisation behaviour of these models. Quantifying generalisation behaviour is non-trivial and there is no clear answer to it. However, we utilise recent efforts that explore the theoretical

understanding of generalisation to study generalisation of SRN in practice.

To inspect the generalisation behaviour in NNs we begin with the shattering experiment [292]. It is a test of whether the network can fit the training data well but not a label-randomised version of it (each image of the dataset is assigned a random label). As $P[y|\mathbf{x}]$ is essentially uninformative because it is a uniformly random distribution, there is no correlation between the labels and the data points. Thus, the best possible test accuracy on this task (the test labels would also be randomised) is approximately 1% on CIFAR100 (10% in CIFAR10). A high training accuracy — which indicates a high generalisation gap (difference between train and test accuracy) can be achieved only by memorising the train data. In these experiments, the training of all models of one architecture are stopped after the same number of epochs, which is double the number of epochs the model is trained for on the clean datasets. Figure 4.4 shows that SRN reduces memorisation on random labels (thus, reduces the estimate of the Rademacher complexity [292]) on CIFAR100. In Figure 4.5, we plot the training accuracy on CIFAR10 and observe a similar result. Importantly, the same model architectures and training strategy were able to achieve high test accuracy for the clean dataset in the previous section.

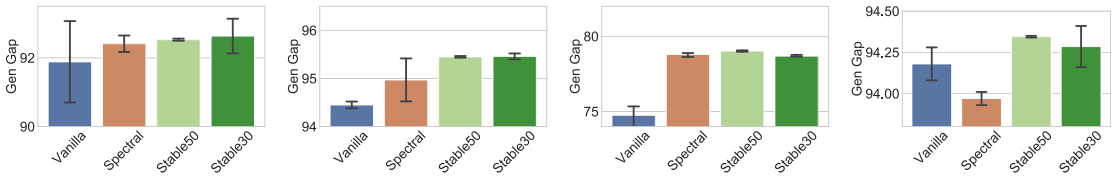
Training accuracy as stopping criterion

We used the number of training epochs as our stopping criterion in the previous experiments. To show that the results hold consistently across different stopping criteria, we use a different stopping criterion here. In particular, we use the training accuracy as a stopping criterion here. For ResNet110, WideResNet-28, Densenet-100, and VGG-19 we use a training accuracy of 99% as a stopping criterion and report the test accuracy when that training accuracy is achieved for the first time. For Alexnet, as SRN-30 never achieves a training accuracy higher than 55%, we use 55% as the stopping criterion and plot the test accuracies in Figure 4.6 for CIFAR100. Our results show that SRN-30 and SRN-50 outperform SN and vanilla consistently. In Figure 4.7, we show similar results for CIFAR10. As AlexNet can achieve higher accuracy for CIFAR10, we use 85% as the stopping criterion for AlexNet on CIFAR10.



(a) ResNet110 (b) WideResNet28 (c) Alexnet (d) Densenet-100 (e) VGG-19

Figure 4.6: Test accuracies on CIFAR100 for clean data using a stopping criterion based on train accuracy. Higher is better.



(a) ResNet110 (b) WideResNet-28 (c) Alexnet (d) Densenet-100

Figure 4.7: Test accuracies on CIFAR10 for clean data using a stopping criterion based on train accuracy. Higher is better.

Non-generalisable settings

So far, we have looked at learning with hyper-parameters like learning rate and weight decay set to values commonly used in training of deep neural networks. However, these hyper-parameters have been adjusted through years of practice. This section looks specifically at settings that are known to be highly non-generalisable — *low learning rate and without weight decay.*

First, we look at a WideResNet-28-10 trained with SRN, SN, and vanilla methods but with a low learning rate of 0.01 and a small weight decay of 5×10^{-4} on randomly labelled CIFAR100 for 50 epochs. The results, shown in Table 4.1, supports our hypothesis that SRN is more robust to random noise than SN or vanilla methods even in this highly-non generalisable setup.

Stable-30	Spectral	Vanilla
29.04	17.24	1.22

Table 4.1: Training error for WideResNet-28-10 on CIFAR100 with randomised labels, low lr= 0.01, and with weight decay. (Higher is better.)

Secondly, we see whether weight decay provides a further edge to generalisation when used in combination with SRN and SN. As shown in Table 4.2, SRN consistently achieves lower generalisation error (by achieving a low train error) both in the presence and the absence of weight decay. Table 4.3 shows it also achieves a higher

clean test accuracy for this setting. Thus SRN generalises better even with a low learning rate and is further benefitted by regularisers like weight decay.

	SRN-50	SRN-30	Spectral (SN)	Vanilla
WD	$\mathbf{12.02 \pm 1.77}$	$\mathbf{11.87 \pm 0.57}$	$\mathbf{11.13 \pm 2.56}$	$\mathbf{10.56 \pm 2.32}$
w/o WD	17.71 ± 2.30	19.04 ± 4.53	17.22 ± 1.94	13.49 ± 1.93

Table 4.2: **Highly non-generalisable setting**. Training error for ResNet-110 on CIFAR100 with randomised labels, low lr= 0.01, and with and without weight decay. (Higher is better.)

	Vanilla	Spectral	Stable-50	Stable-30
W/o WD	69.2 ± 0.5	69 ± 0.1	69.1 ± 0.85	69.3 ± 0.4
With WD	$\mathbf{70.4 \pm 0.3}$	$\mathbf{71.35 \pm 0.25}$	$\mathbf{70.6 \pm 0.1}$	$\mathbf{70.6 \pm 0.1}$

Table 4.3: Clean test accuracy on CIFAR10. The learning configuration corresponds to the non-generalisable settings with a high learning rate.

4.4.3 Empirical evaluation of generalisation behaviour

When all the factors in training (eg. architecture, dataset, optimiser, among others) are fixed, and the only variability is in the normalization (i.e. in SRN vs SN vs Vanilla), the generalisation error can be written as $|\text{Train Err} - \text{Test Err}| \leq \tilde{\mathcal{O}}\left(\sqrt{C_{\text{alg}}/m}\right)$ where $\tilde{\mathcal{O}}(\cdot)$ ignores the logarithmic terms, m is the number of samples in the dataset, and C_{alg} denotes a measure of *sample complexity* for a given algorithm i.e. SRN vs SN vs Vanilla. The lower the value of C_{alg}, the better is the generalisation. This is reminiscent of the basic theorem of generalisation previously discussed in Theorem A. In this section, we measure C_{alg} as a proxy for measuring generalisation.

First, we restate some of the concepts we had previously discussed in Section 3.1.3. The margin of a network, defined in Definition 10 at a data point measures the gap in the confidence of the network between the correct label and the other labels. In the rest of the section, we will treat γ as a random variable depending on the random variables $\mathcal{X}$ and $\mathcal{Y}$.

Definition 10 (Margin)**.** *The margin of a hypothesis h at a data sample $(\mathbf{x}, y)$ is*

defined as

$$\gamma\left(h; \mathbf{x}, y\right) = h\left(\mathbf{x}\right)[y] - \max_{j \neq y} h\left(\mathbf{x}\right)[j]$$

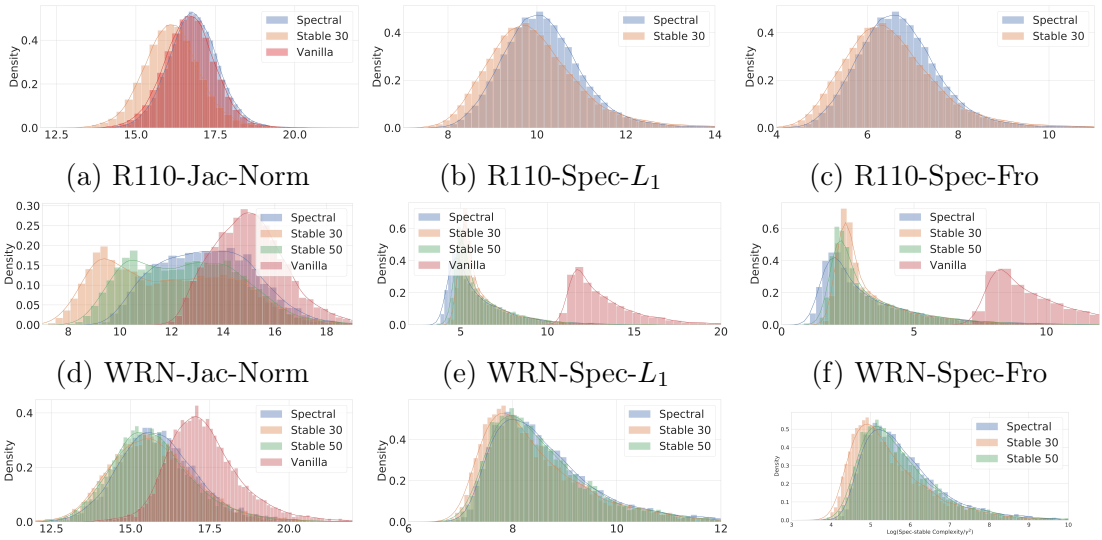
where $h\left(\mathbf{x}\right)[j]$ *indexes the* j^{th} *element of* $h\left(\mathbf{x}\right)$.

We also restate the expressions for C_{alg} that we used previously in Theorem D and Equation (3.10). In addition, we define a new complexity measure from Wei and Ma [269].

- **Spec-Fro:** $\frac{\prod_{i=1}^{L} \|\mathbf{W}_i\|_2^2 \sum_{i=1}^{L} \text{srank}(\mathbf{W}_i)}{\gamma^2}$ [196]. The two quantities used to normalize the margin (γ) are the product of spectral norm i.e. $\prod_{i=1}^{L} \|\mathbf{W}_i\|_2^2$ (or worst case lipschitzness) and the sum of stable rank *i.e.*, $\sum_{i=1}^{L} \text{srank}(\mathbf{W}_i)$ (or an approximate parameter count like rank of a matrix).
- **Spec-L1:** $\frac{\prod_{i=1}^{L} \|\mathbf{W}_i\|_2^2 \left(\sum_{i=1}^{L} \frac{\|\mathbf{W}_i\|_{2,1}^{2/3}}{\|\mathbf{W}_i\|_2^{2/3}}\right)^3}{\gamma^2}$, where $\|.\|_{2,1}$ is the matrix 2-1 norm. As showed by Bartlett et al. [22], Spec-L1 is the spectrally normalized margin, and unlike just the margin, is a good indicator of the generalisation properties of a network.
- **Jac-Norm:** $\sum_{i=1}^{L} \frac{\|\mathbf{h}_i\|_2 \|\mathbf{J}_i\|_2}{\gamma}$ [269], where $\mathbf{h}_i$ is the i^{th} hidden layer and $\mathbf{J}_i = \frac{\partial \gamma}{\partial h_i}$ *i.e.*, the Jacobian of the margin with respect to the i^{th} hidden layer (thus, a vector). Note, Jac-Norm depends on the norm of the Jacobian (local empirical Lipschitz) and the norm of the hidden layers, which are additional data-dependent terms compared to Spec-Fro and Spec-L1.

We evaluate the above-mentioned sample complexity measures on $10,000$ points from the dataset and plot the distribution of its log using a histogram shown in Figure 4.8. The more to the left the histogram is, the smaller is the value of C_{alg} and thus the better is the generalisation capacity of the network.

For better clarity, we report the 90 percentile mark for each of these histograms in Table 4.4. As the plots and the table shows, both SRN and SN produces a much smaller complexity measure than a vanilla network and in 7 out of the 9 cases, SRN is better than SN. The difference between SRN and SN is much more significant in the case of Jac-Norm. As the Jac-Norm depends on the empirical lipschitzness,



(a) R110-Jac-Norm (b) R110-Spec-L_1 (c) R110-Spec-Fro

(d) WRN-Jac-Norm (e) WRN-Spec-L_1 (f) WRN-Spec-Fro

(g) D100-Jac-Norm (h) D100-Spec-L_1 (i) D100-Spec-Fro

Figure 4.8: (log) Sample complexity (C_{alg}) of ResNet-110 (Figures 4.8(a) to 4.8(c)), WideResNet-28-10 (Figures 4.8(d) to 4.8(f)), and Densenet-100 (Figures 4.8(g) to 4.8(i)) quantified using the three measures discussed in this chapter. Left is better. Vanilla is omitted from Figures 4.8(b), 4.8(c), 4.8(h) and 4.8(i) as it is too far to the right. Also, in situations where SRN-50 and SN performed the same, we removed the histogram to avoid clutter.

it provides the empirical validation of our arguments in Section 4.3. *The above*

Model	Algorithm	Jac-Norm	Spec-L_1	Spec-Fro
ResNet-110	Vanilla	17.7	∞	∞
	Spectral (SN)	17.8	10.8	7.4
	SRN-30	**17.2**	**10.7**	**7.2**
WideResNet-28-10	Vanilla	16.2	14.60	11.18
	Spectral (SN)	16.13	7.23	4.5
	SRN-50	15.8	7.3	4.5
	SRN-30	**15.7**	**7.20**	**4.4**
Densenet-100	Vanilla	19.2	∞	∞
	Spectral (SN)	17.8	12.2	9.4
	SRN-50	17.6	12	9.2
	SRN-30	**17.7**	**11.8**	**9.0**

Table 4.4: Values of 90 percentile of log complexity measures from Figure 4.8. Here ∞ refers to the situations where the product of spectral norm blows up. This is the case in deep networks like ResNet-110 and Densenet-100 where the absence of spectral normalization (Vanilla) allows the product of spectral norm to grow arbitrarily large with an increasing number of layers. Lower is better.

experiments indicate that SRN, while providing enough capacity for the standard classification task, is remarkably less prone to memorisation and provides improved generalisation.

4.5 Experiments on a generative modelling setup (SRN-GAN)

In GANs, there is a natural tension between the *capacity* and the *generalisability* of the discriminator. The capacity ensures that if the generated distribution and the data distribution are different, the discriminator can distinguish them. At the same time, the discriminator has to be generalisable, implying, the hypothesis class of discriminators should be simple enough to ensure that the discriminator cannot memorise the dataset. Based on these arguments, we use SRN in the discriminator of GAN which we call SRN-GAN, and compare it against SN-GAN, WGAN-GP, and orthonormal regularisation based GAN (Ortho-GAN).

GAN objective functions In the case of conditional GANs [179], we use the conditional batch normalization [73] to condition the generator and the projection discriminator [181] to condition the discriminator. The dimension of the latent variable for the generator is set to 128 and is sampled from a zero mean and unit variance gaussian distribution. For training the model, we use the hinge loss version of the adversarial loss [163, 257] in all experiments except the experiments with WGAN-GP. The hinge loss version is chosen as it has been shown to consistently give better performance [293, 182]. For training the WGAN-GP model, we use the original loss function as described in Gulrajani et al. [97].

Constructing the empirical lipschitz histogram Along with providing results using evaluation metrics such as Inception score (IS) [227] , FID [114], and Neural divergence score (ND) [98], we use histograms of the empirical Lipschitz constant, *referred to as eLhist* from now onwards, for the purpose of analyses. For a given trained GAN (unconditional), we create $2,000$ pairs of samples, where each pair $(\mathbf{x}_i, \mathbf{x}_j)$ consists of $\mathbf{x}_i$ (randomly sampled from the 'real' dataset) and $\mathbf{x}_j$ (randomly sampled from the generator). Each pair is then passed through the discriminator to compute the empirical lipschitzness based on the pair $\left\|\mathrm{f}(\mathbf{x}_\mathrm{i})-\mathrm{f}(\mathbf{x}_\mathrm{j})\right\|_2 / \left\|\mathbf{x}_\mathrm{i}-\mathbf{x}_\mathrm{j}\right\|_2$, which we then use to create the histogram. In the conditional setting, we sample a class from a discrete uniform distribution and then follow the same approach as described for the

	Algorithm	Inception Score	FID	Intra-FID
Uncond.	Orthonormal[1]	$7.92 \pm .04$	23.8	-
	WGAN-GP	$7.86 \pm .07$	21.7	-
	SN-GAN[1]	$8.22 \pm .04$	20.67	-
	SRN-70-GAN	$\mathbf{8.53} \pm 0.04$	**19.83**	-
	SRN-50-GAN	8.33 ± 0.06	**19.57**	-
Cond.	SN-GAN	$8.71 \pm .04$	16.04[3]	26.24
	SRN-70-GAN	$\mathbf{8.93} \pm 0.12$	**15.92**	**24.01**
	SRN-50-GAN	8.76 ± 0.09	16.89	27.3

Table 4.5: Inception and FID score on CIFAR10.

unconditional setting. Henceforth, we will use the word *histogram* as synonymous to *the histogram of the empirical Lipschitz constant.*

4.5.1 Effect of SRN on Inception Score, FID, and Neural Divergence

Inception Score (IS) of SRN-GANs, SN-GAN, WGAN-GP, and GANs with Orthonormal regularisations on CIFAR10 and CIFAR100 are reported in Tables 4.5 and 4.6 respectively. For measuring the inception score, we generate $50,000$ samples, as is recommended in Salimans et al. [227]. For measuring FID, we use the same setting as Miyato et al. [182] where we sample $10,000$ data points from the training set and compare its statistics with that of $5,000$ generated samples. In addition, we use a recent evaluation metric called Neural divergence score Gulrajani et al. [98] which is more robust to memorisation. The exact set-up for the same is discussed below. In the case of conditional image generation, we also measure Intra-FID [182], which is the mean of the FID of the generator, when it is conditioned over different classes. Let $\text{FID}(\mathcal{G}, c)$ be the FID of the generator $\mathcal{G}$ when it is conditioned on the class $c \in \mathcal{C}$ (where $\mathcal{C}$ is the set of classes), then, Intra $\text{FID}(\mathcal{G}) = \frac{1}{|\mathcal{C}|}\text{FID}(\mathcal{G}, c)$

For CIFAR10, we report results on both the conditional and the unconditional settings and for CIFAR100, we report on the unconditional setting. The results show that SRN-GAN consistently provides a better FID score and an extremely competitive inception score on both: CIFAR10 (both conditional and unconditional

[1]Results are taken from Miyato et al. [182]. The rest of the results in the tables are generated by us.

Model	IS	FID
SN-GAN	**9.04**	23.2
SRN-GAN (Our)	8.85	**19.55**

Table 4.6: Inception and FID score on CIFAR100.

setting) and CIFAR100 (unconditional setting).

In Table 4.7, we compare the Neural Divergence (ND) loss on CIFAR10 and CelebA datasets. Note that ND has been looked at as a metric *more robust to memorisation* than FID and IS in recent works [98, 15]. The exact setting for computing ND is discussed in Appendix B. We essentially report the loss incurred by a *fresh* classifier trained to discriminate the generator distribution and the data distribution. Thus higher the loss, the better are the generated images. As evident from Table 4.7, SRN-GAN provides better ND scores on both datasets.

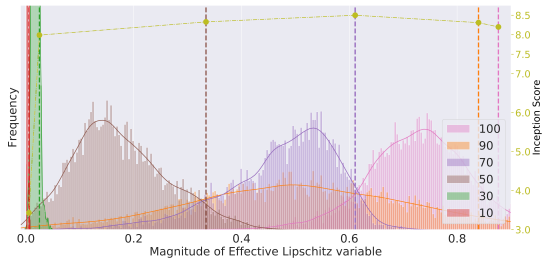
Model	CIFAR10	CelebA
SN-GAN	10.69	0.36
SRN-GAN (Our)	**11.97**	**0.64**

Table 4.7: Neural Discriminator Loss (Higher the better).

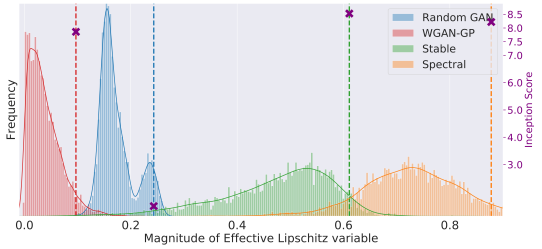
4.5.2 Effect of SRN on eLhist

We construct eLhist for unconditional GANs on CIFAR10 in Figure 4.9(a). As the plot shows, lowering the value of c (aggressive reduction in the stable rank) shifts the histogram towards zero, implying a lower empirical Lipschitz constant. This validates the arguments provided in Section 4.3.1. Lowering c also improves the inception score. However, an extreme reduction in the stable rank ($c = 0.1$) dramatically collapses the histogram to zero and also drops the inception score significantly. This is because, at $c = 0.1$, the capacity of the discriminator is reduced to the point that it is not able to learn to differentiate between the real and the fake samples anymore.

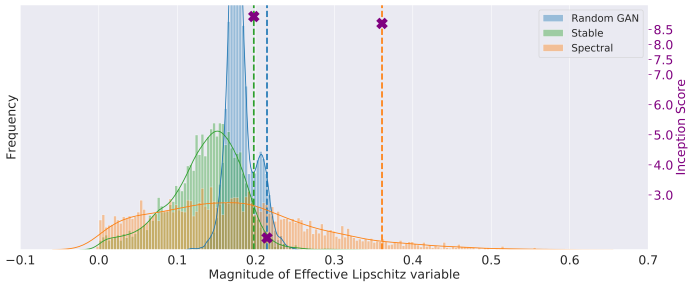
In addition, in Figure 4.9(b), we provide eLhist for comparing different approaches. Random-GAN is a randomly initialised GAN with the same architecture as SRN-GAN. As expected, Random-GAN has a low empirical Lipschitz constant



(a) Unconditional GAN



(b) Unconditional GAN

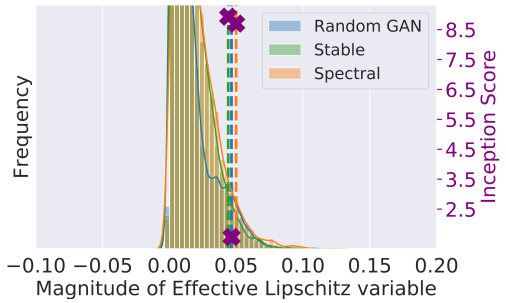


(c) Conditional GAN with projection discriminator

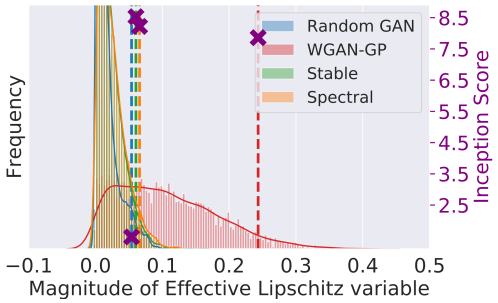
Figure 4.9: **eLhist** for unconditional and conditional GANs on CIFAR10. Dashed vertical lines represent $95th$ percentile. Solid circles and crosses represent the *inception score* for each histogram. Figure 4.9(a) shows SRN-GAN for different stable rank constraints (*e.g.* 90 implies $c = 0.9$). Figure 4.9(b) compares various approaches for unconditional GANs. Figure 4.9(c) compares various approaches for conditional GAN setting. Random-GAN represents random initialisation (no training). For SRN-GAN, we use $c = 0.7$.

and an extremely poor inception score. Unsurprisingly, WGAN-GP has a lower empirical lipschitz constant than Random-GAN, due to its explicit constraint on the Lipschitz constant, while providing a higher inception score. On the other hand, SRN-GAN, by virtue of its softer constraints on the Lipschitz constant, trades off a higher Lipschitz constant than WGAN-GP for a better inception score—highlighting the flexibility provided by SRN. We show results on the conditional GAN setup using the projection discriminator from Miyato and Koyama [181] in Figure 4.9(c).

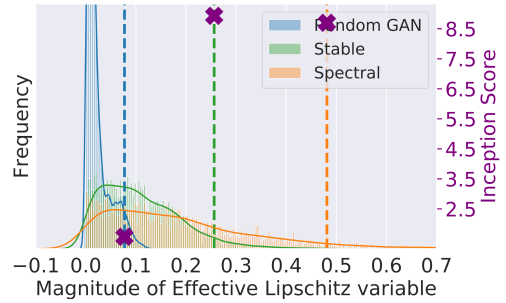
For the purpose of analysis, Figures 4.10(b) and 4.10(d) shows eLhist for pairs where each sample either comes from the true data or the generator, and we observe a similar trend. To verify the same results hold in the conditional setup, we show comparisons for GANs with projection discriminator [181] in Figures 4.10(a) and 4.10(c), and also observe a similar trend.



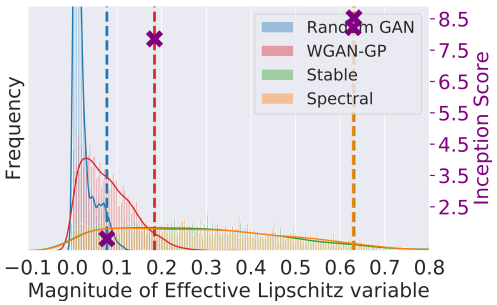
(a) Conditional GAN with projection discriminator.



(b) Unconditional GAN setting.



(c) Conditional GAN with projection discriminator



(d) Unconditional GAN setting.

Figure 4.10: Figures 4.10(a) and 4.10(b) plots the **eLhist** of the discriminator for pairs of samples drawn from the generator on CIFAR10. Figures 4.10(c) and 4.10(d) plots the **eLhist** of the discriminator for pairs of samples drawn from the real distribution on CIFAR10.

5. Impact of Label Noise and Representation Learning on Adversarial Robustness

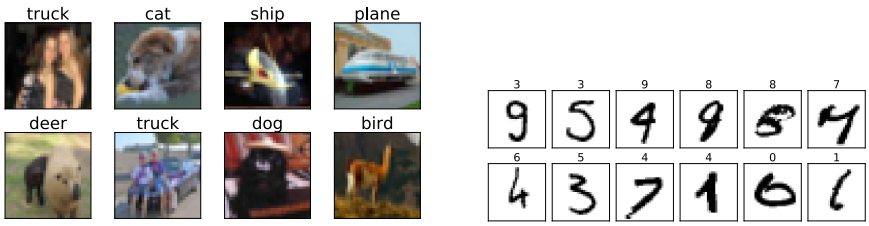
Despite being successful in a wide range of tasks e.g. in computer vision, [146, 289, 109, 217] and natural language processing [96, 266], machine learning algorithms have been shown to be highly vulnerable to small adversarial perturbations that are otherwise imperceptible to the human eye [62, 29, 248, 93, 43, 205, 186]. This vulnerability poses serious security concerns when these models are deployed in real-world tasks (cf. [207, 235, 113, 160]). A large body of research has been devoted to crafting defences to protect neural networks from adversarial attacks (e.g. [93, 204, 256, 172, 294]). However, as we discussed in Section 3.2 these defences have usually been broken by future attacks [18, 255]. This arms race between attacks and defences suggests that creating a truly robust model would require a deeper understanding of the source of this vulnerability.

Our goal in this chapter is not to propose new defences, but to provide better answers to the question: what causes the adversarial vulnerability? In doing so, we also try to understand how existing methods designed to achieve adversarial robustness overcome some of these hurdles pointed out by our work. In Chapter 4, we discussed the memorisation of label noise in the context of generalisation. In this chapter, we will look at the impact of memorisation on adversarial vulnerability.

We identify two sources of adversarial vulnerability that, to the best of our knowledge, have not been properly studied before: a) memorisation of label noise, and b) *improper* representation learning.

5.1 Theoretical result on the impact of memorising label noise

Starting with the celebrated work of Zhang et al. [292] it has been observed that neural networks trained with SGD are capable of memorising large amounts of label



CIFAR10 MNIST

Figure 5.1: Label noise in CIFAR10 and MNIST. The text above the image indicates the training set label.

noise. Recent theoretical work (e.g. [162, 26, 25, 108, 27, 28, 23, 190, 47]) has also sought to explain why fitting training data perfectly does not lead to a large drop in test accuracy, as the classical notion of overfitting might suggest. This is commonly referred to as *memorisation* or *interpolation*. We show through simple theoretical models, as well as experiments on standard datasets, that there are scenarios where label noise causes significant adversarial vulnerability, even when high natural (test) accuracy can be achieved. Surprisingly, we find that label noise is not at all uncommon in datasets such as MNIST and CIFAR-10 (see Figure 5.1).

We develop a simple theoretical framework to demonstrate how overfitting, even very minimal, label noise causes significant adversarial vulnerability. First, we recall the definitions of expected error, which we refer to as *Natural error* in this chapter, and adversarial error.

Definition 19 (Natural Error)**.** *For any distribution $\mathcal{D}$ defined over $(\mathbf{x}, y) \in \mathbb{R}^d \times \{0,1\}$ and any binary classifier $f : \mathbb{R}^d \to \{0,1\}$, the* natural *error is*

$$\mathcal{R}(f; \mathcal{D}) = \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}} \left[f(\mathbf{x}) \neq y \right], \tag{5.1}$$

Definition 17 (ℓ_p Adversarial Error)**.** *For any distribution $\mathcal{D}$ defined over $(\mathbf{x}, y) \in \mathbb{R}^d \times \mathcal{Y}$, any classifier $h : \mathbb{R}^d \to \mathcal{Y}$, and any $\gamma > 0$, the* γ-adversarial *error is*

$$\mathcal{R}_{\mathrm{Adv},\gamma}(h; \mathcal{D}) = \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\exists \mathbf{z} \in \mathcal{B}_\gamma(\mathbf{x}) \, ; h(\mathbf{z}) \neq y \right], \tag{3.12}$$

where $\mathcal{B}_\gamma^p(\mathbf{x})$ is the ℓ_p ball of radius $\gamma \geq 0$ around $\mathbf{x}$ under the ℓ_p norm.

The following result provides a sufficient condition under which even a small

amount of label noise causes any classifier that fits the training data perfectly to have significant adversarial error. Informally, Theorem 2 states that if the data distribution has significant probability mass in a union of (a relatively small number of, and possibly overlapping) balls, each of which has roughly the same probability mass (see Equation (5.2)), then even a small amount of label noise renders this entire region vulnerable to adversarial attacks to classifiers that fit the training data perfectly.

Theorem 2 (Overfitting Label Noise Causes Adversarial Vulnerability)**.** *Let c be the target classifier, and let $\mathcal{D}$ be a distribution over $(\mathbf{x}, y)$, such that $y = c(\mathbf{x})$ in its support. Using the notation $\mathbb{P}_{\mathcal{D}}[A]$ to denote $\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}[\mathbf{x} \in A]$ for any measurable subset $A \subseteq \mathbb{R}^d$, suppose that there exists $c_1 \geq c_2 > 0$, $\rho > 0$, and a finite set $\zeta \subset \mathbb{R}^d$ satisfying*

$$\mathbb{P}_{\mathcal{D}}\left[\bigcup_{\mathbf{s}\in\zeta} \mathcal{B}_\rho^p(\mathbf{s})\right] \geq c_1 \quad \textit{and} \quad \forall \mathbf{s} \in \zeta,\ \mathbb{P}_{\mathcal{D}}\left[\mathcal{B}_\rho^p(\mathbf{s})\right] \geq \frac{c_2}{|\zeta|} \tag{5.2}$$

where $\mathcal{B}_\rho^p(\mathbf{s})$ represents an ℓ_p-ball of radius ρ around $\mathbf{s}$. Further, suppose that each of these balls contains points from a single class i.e. for all $\mathbf{s} \in \zeta$, for all $\mathbf{x}, \mathbf{z} \in \mathcal{B}_\rho^p(\mathbf{s}) : c(\mathbf{x}) = c(\mathbf{z})$.

Let $\mathcal{S}_m$ be a dataset of m i.i.d. samples drawn from $\mathcal{D}$, which subsequently has each label flipped independently with probability η. For any classifier f that perfectly *fits the training data $\mathcal{S}_m$ i.e. $\forall\ \mathbf{x}, y \in \mathcal{S}_m, f(\mathbf{x}) = y$, $\forall \delta > 0$ and $m \geq \frac{|\zeta|}{\eta c_2} \log\left(\frac{|\zeta|}{\delta}\right)$, with probability at least $1 - \delta$, $\mathcal{R}_{\mathrm{Adv},2\rho}(f; \mathcal{D}) \geq c_1$.*

The goal is to find a relatively small set ζ that satisfies the condition as this will mean that even for modest sample sizes, the trained models have significant adversarial error. We remark that it is easy to construct concrete instantiations of problems that satisfy the conditions of the theorem, e.g. when each class is represented by a spherical (truncated) Gaussian with radius ρ and the classes are well-separated, the distribution satisfies Equation (5.2). The main idea of the proof is that there is sufficient probability mass for points that are within distance 2ρ of a training datum that was mislabelled. We note that the generality of the result, namely that *any* classifier (including neural networks) that fits the training data must be vulnerable irrespective of its structure, requires a result like Theorem 2.

For instance, one could construct the classifier h, where $h(\mathbf{x}) = c(\mathbf{x})$, if $(\mathbf{x}, b) \notin \mathcal{S}_m$ for $b = 0, 1$, and $h(\mathbf{x}) = y$ if $(\mathbf{x}, y) \in \mathcal{S}_m$. Note that the classifier h agrees with the target c on *every* point of $\mathbb{R}^d$ except the mislabelled training examples, and as a result, these examples are the only source of vulnerability. Also note that if the classifier h were smoother in the vicinity of the memorised misclassified point, then the vulnerability of the classifier would actually increase. This is surprising as normally smoother classifiers are thought to be less vulnerable to adversarial attacks.

Proof of Theorem 2. From Equation (5.2), for any ζ and $s \in \zeta$,

$$\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\mathbf{x} \in \mathcal{B}_\rho\left(s\right)\right] \geq \frac{c_2}{|\zeta|}$$

As the sampling of the point and the injection of label noise are independent events,

$$\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\mathbf{x} \in \mathcal{B}_\rho\left(s\right) \wedge \mathbf{x} \text{ gets mislabelled}\right] \geq \frac{c_2\eta}{|\zeta|}$$

Thus,

$$\begin{aligned}\mathbb{P}_{\mathcal{S}_m\sim\mathcal{D}^m}\left[\exists\left(\mathbf{x}, y\right) \in \mathcal{S}_m : \mathbf{x} \in \mathcal{B}_\rho\left(s\right) \wedge \mathbf{x} \text{ is mislabelled}\right] &\geq 1 - \left(1 - \frac{c_2\eta}{|\zeta|}\right)^m \\ &\geq 1 - \exp\left(\frac{-c_2\eta m}{|\zeta|}\right) \quad \text{(By Inequality 1)}\end{aligned}$$

Substituting $m \geq \frac{|\zeta|}{\eta c_2}\log\left(\frac{|\zeta|}{\delta}\right)$ and applying the union bound (Inequality 2) over all $s \in \zeta$, we get

$$\mathbb{P}_{\mathcal{S}_m\sim\mathcal{D}^m}\left[\forall s \in \zeta,\ \exists\left(\mathbf{x}, y\right) \in \mathcal{S}_m : \mathbf{x} \in \mathcal{B}_\rho\left(s\right) \wedge \mathbf{x} \text{ is mislabelled}\right] \geq 1 - \delta \tag{5.3}$$

As for all $\mathbf{s} \in \mathbb{R}^d$ and $\forall \mathbf{x}, \mathbf{z}, \in \mathcal{B}_\rho^p\left(\mathbf{s}\right),\ \left\|\mathbf{x} - \mathbf{z}\right\|_p \leq 2\rho$, we have that

$$\begin{aligned}\mathcal{R}_{\text{Adv},2\rho}(f; \mathcal{D}) &= \mathbb{P}_{\mathcal{S}_m\sim\mathcal{D}^m}\left[\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\exists \mathbf{z} \in \mathcal{B}_{2\rho}\left(\mathbf{x}\right)\ \wedge y \neq f\left(\mathbf{z}\right)\right]\right] \\ &\geq \mathbb{P}_{\mathcal{S}_m\sim\mathcal{D}^n}\left[\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\mathbf{x} \in \bigcup_{s\in\zeta}\mathcal{B}_\rho^p\left(s\right) \wedge \left\{\exists \mathbf{z} \in \mathcal{B}_{2\rho}\left(\mathbf{x}\right) : y \neq f\left(\mathbf{z}\right)\right\}\right]\right]\end{aligned}$$

$$
\begin{aligned}
&\overset{(1)}{=} \mathbb{P}_{\mathcal{S}_m \sim \mathcal{D}^n}\left[\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\exists \mathbf{s} \in \zeta : \mathbf{x} \in \mathcal{B}_\rho^p(s) \wedge \{\exists \mathbf{z} \in \mathcal{B}_{2\rho}(\mathbf{x}) : y \neq f(\mathbf{z})\}\right]\right] \\
&\geq \mathbb{P}_{\mathcal{S}_m \sim \mathcal{D}^m}\left[\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\exists \mathbf{s} \in \zeta : \mathbf{x} \in \mathcal{B}_\rho^p(s) \wedge \{\exists \mathbf{z} \in \mathcal{B}_{\rho}(\mathbf{s}) : y \neq f(\mathbf{z})\}\right]\right] \\
&\overset{(2)}{=} \mathbb{P}_{\mathcal{S}_m \sim \mathcal{D}^m}\left[\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\exists \mathbf{s} \in \zeta : \mathbf{x} \in \mathcal{B}_\rho^p(s) \wedge \{\exists \mathbf{z} \in \mathcal{B}_{\rho}(\mathbf{s}) : c(\mathbf{z}) \neq f(\mathbf{z})\}\right]\right] \\
&\overset{(3)}{=} \mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\mathbf{x} \in \bigcup_{s\in\zeta} \mathcal{B}_\rho^p(s)\right] \quad \text{w.p. at least } 1-\delta \\
&\geq c_1 \quad \text{w.p. } 1-\delta
\end{aligned}
$$

where c is the true concept for the distribution $\mathcal{D}$.

Equality (1) follows from the fact that the event $\mathbf{x} \in \bigcup_{s\in\zeta} \mathcal{B}_\rho^p(s)$ is equivalent to the event that there exists $i \in \{1, .., |\zeta|\}$ such that $\mathbf{x} \in \mathcal{B}_\rho^p(s_i)$. Equality (2) follows from the assumptions that each of the balls around $\mathbf{s} \in \zeta$ is pure in their labels. Equality (3) follows from 5.3 by using the fact that $\mathbf{x}$ is guaranteed to exist in the ball around $\mathbf{s}$ and be mislabelled with a probability of at least $1-\delta$. To see how, replace the $\mathbf{x}$ in Assumption 5.3 with $\mathbf{z}$. The last inequality follows from the first Assumption in 5.2. □

There are a few things to note about Theorem 2.

1. First, the lower bound on adversarial error applies to any classifier f that fits the training data $\mathcal{S}_m$ perfectly and is agnostic to the type of model f is.

2. Second, for a given c_1, there may be multiple ζs that satisfy the bounds in Equation (5.2) and the adversarial risk holds for all of them. Thus, the smaller the value of $|\zeta|$ the smaller the size of the training data it needs to fit which can be obtained by simpler classifiers.

3. Third, if the distribution of the data is such that it is concentrated around some points then for a fixed c_1, c_2, a smaller value of ρ would be required to satisfy Equation (5.2) and thus a weaker adversary (smaller perturbation budget 2ρ) can cause a much larger adversarial error.

Remark 4. *We remark that an interesting future direction of research would be to understand the relation between ζ, c_1, c_2, and ρ for various common distributions*

and identify which distributions are more vulnerable to adversarial attacks than others. Further, it would be interesting to investigate whether the lower bound on adversarial risk can be further increased by removing the hypothesis-agnostic nature of Theorem 2. This would allow us to understand what kind of distributions and machine learning models are fundamentally more vulnerable to adversarial attacks due to memorisation of label noise.

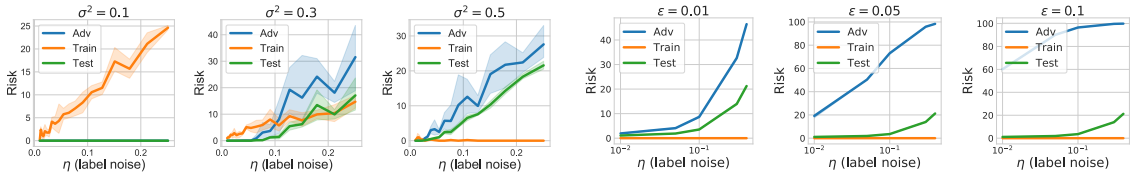
In practice, classifiers exhibit much greater vulnerability than purely arising from the presence of memorised noisy data. However, experiments in Section 5.2 shows how label noise causes vulnerability in a toy MNIST model, the full MNIST and CIFAR10 for a variety of architectures. This indicates that while removing label noise is not sufficient to show adversarial robustness for interpolating classifiers, it is a necessary condition.

5.2 Experimental results on the impact of memorising label noise

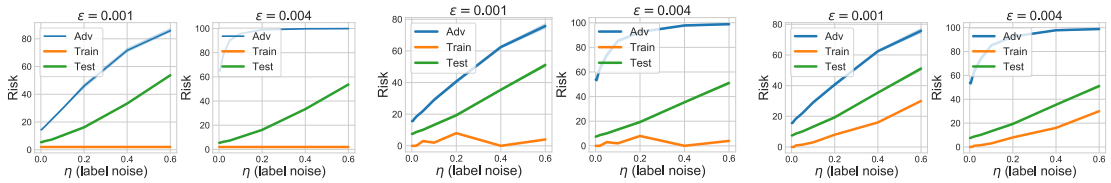
This section will look at empirical results on synthetic data, inspired by the theory and on the standard datasets: MNIST [156] and CIFAR10 [145], that shows the impact of memorising label noise on adversarial vulnerability.

5.2.1 Memorisation of label noise hurts adversarial accuracy

Experiments on toy-MNIST We design a simple binary classification problem, *toy-MNIST*, and show that when fitting a complex classifier on a training dataset with label noise, adversarial vulnerability increases with the amount of label noise and that this vulnerability is caused by the label noise. The problem is constructed by selecting two images from MNIST: one "0" and one "1". Each training/test example is generated by selecting one of these images and adding i.i.d. Gaussian noise sampled from $\mathcal{N}(0, \sigma^2)$ for some $\sigma > 0$. We create a training dataset of 4000 samples by sampling uniformly from either class. Finally, η fraction of the training data is chosen randomly and its labels are flipped. We train a neural network with four fully connected layers followed by a softmax layer and minimise



(a) Toy-MNIST , $\epsilon = \frac{64}{255}$ (b) Full-MNIST



(c) ResNet18 (CIFAR10) (d) DenseNet121 (CIFAR10) (e) VGG19 (CIFAR10)

Figure 5.2: Adversarial Error increases with increasing label noise η. The shaded region indicates a 95% confidence interval. The absence of a shaded region indicates that it is invisible due to low variance.

the cross-entropy loss using an SGD optimiser until the training error becomes zero. Then, we attack this network with a *strong* ℓ_∞ PGD adversary (discussed in Section 3.2.2) [172] with $\epsilon = \frac{64}{255}$ for 400 steps with a step size of 0.01.

In Figure 5.2(a), we plot the adversarial error, natural test error and training error as the amount of label noise (η) varies, for three different values of sample variance (σ^2). For low values of σ^2 ($\sigma^2 = 0.1$), the training data from each class are all concentrated around the same point; as a result, these models are unable to memorise the label noise and the training error is high. In this case, over-fitting label noise is impossible and the test error, as well as the adversarial error, is low. This does not contradict Theorem 2, which requires zero training error on the mislabelled dataset. However, as σ^2 increases to $\sigma^2 = 0.5$, the neural network is flexible enough to use the "noise component" to extract features that allow it to memorise label noise and fit the training data perfectly. This brings the training error down to zero while causing the test error to increase, and the adversarial error even more so. This is in line with Theorem 2.

Remark 5. *The case when $\sigma^2 = 0.3$ is particularly interesting; when the label noise is low and the training error is high, there is no overfitting and the test error and the adversarial error is zero. When the network starts memorising label noise (i.e. train error gets lesser than label noise), test error remains very low but adversarial error increases rapidly.*

Experiments on the full MNIST and CIFAR10 dataset We perform a similar experiment on the full MNIST dataset trained on a 4-layered Convolutional Neural Network. The model architecture consists of four convolutional layers, followed by two fully connected layers. The first four convolutional layers have $32, 64, 128$, and 256 output filters with kernels of width $3, 4, 3$, and 3 respectively. The two fully connected layers have a width of 1024. The network is optimised with SGD with a batch size of 128 and an initial learning rate of 0.1 for a total of 60 epochs. The learning rate is decreased to 0.01 after 50 epochs. For varying values of η, we assign a uniformly randomly label to a randomly chosen η fraction of the training data. We compute the natural test accuracy and the adversarial test accuracy on a clean test-set with no label noise for when the network is attacked with an ℓ_∞ bounded PGD adversary for varying perturbation budget ϵ, with a step size of 0.01 and for 20 steps and plot the results in Figure 5.2(b). We repeat the same experiment for CIFAR10 with a DenseNet121 [120], ResNet18 [110], and VGG19 [239] to test the phenomenon across multiple state-of-the-art architectures and plot the results in Figures 5.2(c) to 5.2(e). Please refer to Appendix B for more details on the architectures and datasets. The results on both datasets show that the effect of over-fitting label noise on adversarial error is even more clearly visible here; for the same PGD adversary, the adversarial error jumps upwards sharply with increasing label noise, while the growth of natural test error is much slower. This confirms the hypothesis that benign overfitting may not be so benign when it comes to adversarial error.

5.2.2 Representations of label noise and adversarial examples

For the toy-MNIST problem, we plot a 2-d projection (using PCA) of the learned representations (activations before the last layer) at various stages of training in Figure 5.3. We remark that the simplicity of the data model ensures that even a 1-d PCA projection suffices to perfectly separate the classes when there is no label noise; however, the representations learned by a neural network in the presence of noise may be very different! We highlight two key observations:

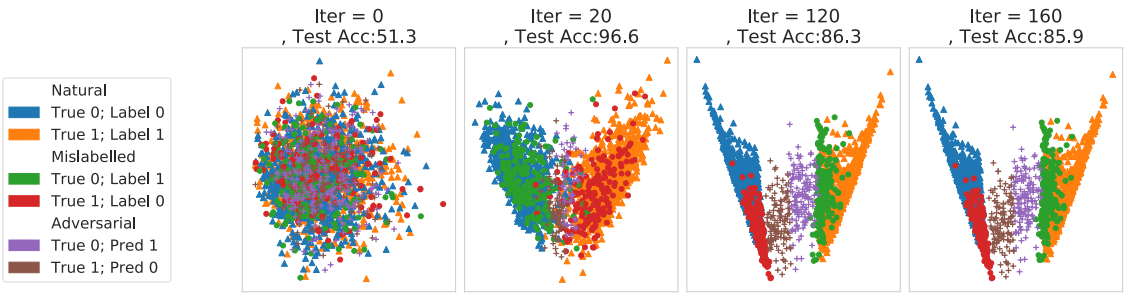


Figure 5.3: Two dimensional PCA projections of the original correctly labelled (blue and orange), original mislabelled (green and red), and adversarial examples (purple and brown) at different stages of training. The correct label for *True 0* (blue), *Noisy 0* (green), *Adv 0* (purple +) are the same i.e. 0 and similar for the other class.

1. The bulk of adversarial examples ("+"-es) are concentrated around the mislabeled training data ("○"-es) of the opposite class. For example, the purple +-es (Adversarially perturbed: True: 0, Pred:1) are very close to the green ○-es (Mislabelled: True:0, Pred: 1). This provides empirical validation for the hypothesis that if there is a mislabeled data point in the vicinity that has been fit by the model, an adversarial example is created by moving towards that data point as predicted by Theorem 2.

2. The mislabeled training data take longer to be fit by the classifier. For example, by iteration 20, the network learns a fairly good representation and classification boundary that correctly fits the clean training data (but not the noisy training data). At this stage, the number of adversarial examples is much lower as compared to Iteration 160, by which point the network has completely fit the noisy training data. Thus early stopping helps in avoiding *memorising* the label noise, and consequently also reduces adversarial vulnerability. Early stopping has indeed been used as a defence in quite a few recent papers in the context of adversarial robustness [277, 112], as well as learning in the presence of label-noise [161]. Our work shows *why* early stopping may reduce adversarial vulnerability by avoiding fitting noisy training data.

5.2.3 Robust training avoids memorisation of rare examples

Robust training methods like AT [172] and TRADES [294] are commonly used data-augmentation based techniques to increase the adversarial robustness of deep

ϵ	Train-Acc. (%)	Test-Acc (%)
0.0	99.98	95.25
0.25	97.23	92.77
1.0	86.03	81.62

Table 5.1: Train and Test Accuracies on Clean Dataset for ResNet-50 models trained using ℓ_2 adversaries

neural networks. However, it has been pointed out that this comes at a cost to clean accuracy [214, 259]. When trained with these methods, both the training and test accuracy (on clean data) for commonly used deep learning models drops with increasing strength of the PGD adversary used in the adversarial training (see Table 5.1). In this section, we provide evidence to show that robust training avoids memorisation of label noise and this also results in the drop of clean train and test accuracy. Before going further, we will first describe how we measure memorisation and related concepts. We borrow these two concepts from Zhang and Feldman [291] who measure the label memorisation phenomenon using two related measures: *memorisation* and *influence*.

Memorisation or Self-Influence: Self influence of an example $(\mathbf{x}_i, y_i)$ for a dataset $\mathcal{S}$ and learning algorithm $\mathcal{A}$ (including model, optimiser etc) can be defined as how unlikely it is for the model learnt by $\mathcal{A}$ to be correct on $(\mathbf{x}_i, y_i)$ if the training dataset $\mathcal{S}$ does not contain $(\mathbf{x}_i, y_i)$ compared to if $\mathcal{S}$ contains $(\mathbf{x}_i, y_i)$. It can be formalised as follows which is borrowed from Eq (1) in Zhang and Feldman [291]

Memorisation by $\mathcal{A}$ on example $(x_i, y_i) \in \mathcal{S}$ is measured as

$$\text{mem}(\mathcal{A}, \mathcal{S}, i) := \mathbb{P}_{h \sim \mathcal{A}(\mathcal{S})}[h(x_i) = y_i] - \mathbb{P}_{h \sim \mathcal{A}(\mathcal{S}^{\setminus i})}[h(x_i) = y_i]$$

where $\mathcal{S}^{\setminus i}$ denotes the dataset $\mathcal{S}$ with (x_i, y_i) removed, $h \sim \mathcal{A}(\mathcal{S})$ denotes the model h obtained by training using algorithm $\mathcal{A}$ (which includes the model architecture) on the dataset $\mathcal{S}$ and the probability is taken over the randomisation inherent in the training algorithm $\mathcal{A}$.

Influence of a training example on a test example: Given a training example $(\mathbf{x}_i, y_i)$, a test example $\left(\mathbf{x}'_j, y'_j\right)$, a training dataset $\mathcal{S}$ and a learning algorithm $\mathcal{A}$,

the influence of $(\mathbf{x}_i, y_i)$ on $\left(\mathbf{x}'_j, y'_j\right)$ measures the probability that $\left(\mathbf{x}'_j, y'_j\right)$ would be classified correctly if the training set $\mathcal{S}$ does not contain $(\mathbf{x}_i, y_i)$ compared to if it does. This can be defined as follows which is borrowed from Eq 2 in Zhang and Feldman [291]. Using a similar notation as memorisation, the influence of (x_i, y_i) on (x'_j, y'_j) for the learning algorithm $\mathcal{A}$ with training dataset $\mathcal{S}$ can be measured as

$$\text{infl}(\mathcal{A}, \mathcal{S}, (x_i, y_i), (x'_j, y'_j)) := \mathbb{P}_{h\sim\mathcal{A}(\mathcal{S})}[h(x'_j) = y'_j] - \mathbb{P}_{h\sim\mathcal{A}(\mathcal{S}^{\setminus i})}[h(x'_j) = y'_j]$$

Robust training ignores label noise Figure 5.1 shows that label noise is not uncommon in standard datasets like MNIST and CIFAR10. In fact, upon closely monitoring the misclassified training set examples for both AT and TRADES, we found that neither AT nor TRADES predicts correctly on the training set labels for any of the examples identified in Figure 5.1, all examples that have a wrong label in the training set, whereas natural training does. Thus, in line with Theorem 2, robust training methods ignore fitting noisy labels.

We also observe this in a synthetic experiment on the full MNIST dataset where we assigned random labels to 15% of the dataset. A naturally trained CNN model achieved 100% train accuracy on this dataset whereas an adversarially trained model (standard setting with $\epsilon = 0.3$ for 30 steps) misclassified 997 examples in the training set after the same training regime. Out of these 997 samples, 994 examples belonged to that 15% of the examples that were mislabelled in the dataset.

Robust training ignores rare examples Certain examples in the training set belong to rare sub-populations (eg. a special kind of cat) and this sub-population is sufficiently distinct from the rest of the examples of that class in the training dataset (other cats in the dataset). Next, we show that though ignoring rare samples possibly helps in adversarial robustness, it hurts the natural test accuracy. We hypothesise that one of the effects of robust training is to not *memorise rare examples*, which would otherwise be memorised by a naturally trained model. As Feldman [80] points out, *if these sub-populations are very infrequent in the training dataset, they are indistinguishable from data points with label noise with the difference be-*

ing that examples from that sub-population are also present in the test-set. Natural training by *memorising* those rare training examples reduces the test error on the corresponding test examples. Robust training, by not memorising these rare samples (and label noise), achieves better robustness but sacrifices the test accuracy on the test examples corresponding to those training points.

Experiments on MNIST, CIFAR10, and ImageNet We visually demonstrate this effect in Figure 5.4 with examples from CIFAR10, MNIST, and ImageNet and then provide more statistical evidence using the notions of memorisation score and influence [291] in Figure 5.5. Each pair of images contains a misclassified (by robustly trained models) test image and the misclassified training image "responsible" for it. Importantly both of these images were correctly classified by a naturally trained model. Visually, it is evident that the training images are extremely similar to the corresponding test image. Inspecting the rest of the training set, they are also very different from other images in the training set. We can thus refer to these as rare sub-populations.

We found the images in Figure 5.4 by manually searching for each test image, the training image that is misclassified and is visually close to it. Our search space was shortened with the help of the influence scores of each training image on the test image. We searched in the set of top-10 most influential misclassified train images for each misclassified test image. The model used for Figure 5.4 is a) an AT ResNet50 model for CIFAR10 with ℓ_2-adversary with an $\epsilon = 0.25$, b) a model trained with TRADES for MNIST with $\lambda = \frac{1}{6}$ and $\epsilon = 0.3$, and c) and AT ResNet50 model for Imagenet with ℓ_2 adversary with $\epsilon = 3.0$. Zhang and Feldman [291] provided us the with the memorisation scores for each image in CIFAR10 as well as the influence score of each training image on each test image for each class in CIFAR-10. High Influence pairs of Imagenet were obtained from `https://pluskid.github.io/influence-memorization/`. This was used to obtain the figures for the Imagenet dataset in Figure 5.4.

A precise notion of measuring if a sample is *rare* is through the concept of self-influence or memorisation. Self-influence for a *rare example*, that is unlike other

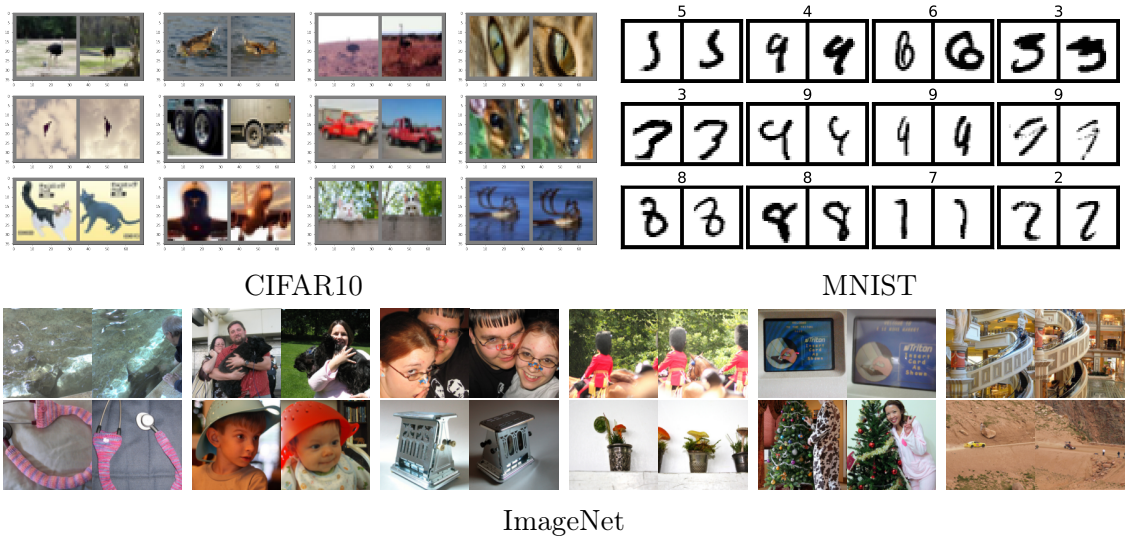
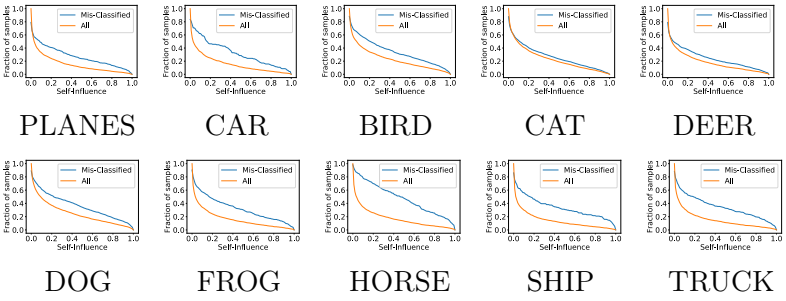


Figure 5.4: Each pair is a training (left) and test (right) image misclassified by the adversarially trained model. They were both correctly classified by the naturally-trained model.

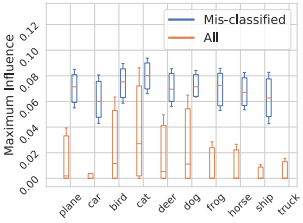
examples of that class, will be high as the rest of the dataset will *not* provide relevant information that will help the model to correctly predict on that particular example. In Figure 5.5(a), we show that the self-influence of training samples that were misclassified by adversarially trained models but correctly classified by a naturally trained model is higher compared to the distribution of self-influence on the entire train dataset. In other words, it means that the self-influence of the training examples misclassified by the robustly trained models is larger than the average self-influence of (all) examples belonging to that class. This supports our hypothesis that adversarial training excludes fitting these rare (or ones that need to be memorised) samples.

The notion that certain test examples were not classified correctly due to a particular training example not being classified correctly is measured by the *influence* a training image has on the test image (c.f. definition 3 in Zhang and Feldman [291]). We obtained the influence of each training image on each test image for that class from Zhang and Feldman [291] and the training images in Figure 5.4 has a disproportionately higher influence on the corresponding test image compared to influences of other train-test image pairs in CIFAR10.

In Figure 5.5(b), we show that the influence of training images are higher on test images that are misclassified by adversarially trained models as compared to an average test image from the dataset. In other words, this means that adversarially



(a) Fraction of train points that have a self-influence greater than s is plotted versus s.



(b) Distribution of the influence of training point on all test points compared to the distribution of influence on test points mis-classified by adversarially trained points.

Figure 5.5: The blue represents the points misclassified by an adversarially trained model. The orange represents the distribution for all points in the dataset (of the concerned class for CIFAR10).

trained models misclassify test examples that are being heavily influenced by some particular training example. As we saw in Figure 5.5(a), AT models do not memorise atypical train examples; consequently, they misclassify test examples that are heavily influenced by those atypical train examples (visualised in Figure 5.4). This confirms our hypothesis that the loss in test accuracy of robustly trained models is due to test images that are *rare* and thus have a particularly high influence from a training image.

Remark 6. *We remark that this behaviour of AT and TRADES seemingly induces a tradeoff between natural test error and adversarial error in the presence of label noise. However, this is under the assumption that the learning algorithm cannot distinguish between examples with label noise and atypical examples. Our experiments indicate that AT and TRADES is unable to distinguish between them. However, if an algorithm can distinguish between them then this specific kind of tradeoff should not arise although there might still be tradeoffs for other reasons.*

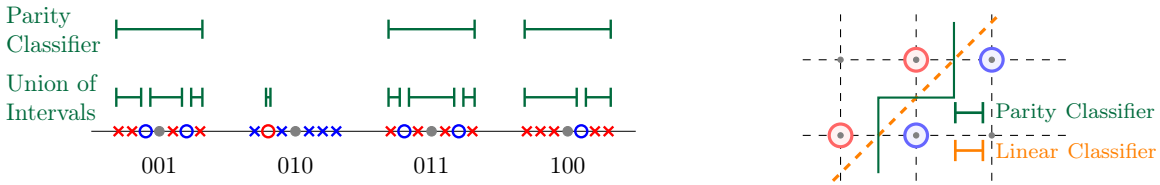
This drop in test accuracy due to not memorising rare or atypical samples is not specific to AT and TRADES but seen in other learning algorithms, that avoid memorisation, as well. In particular, this has been observed in differentially private training [20] and sparse training [117] as well.

5.3 Theoretical results on the impact of representation learning

A large portion of the success of modern machine learning methods has largely been made possible by incorporating proper inductive biases, and that has helped achieve better generalisation and accelerate optimisation. In this section, we will look at the harmful impacts of choosing the incorrect inductive bias. In particular, we look at *improper* representation learning by way of incorrect inductive biases as a source of adversarial vulnerability.

Recent works [259, 294] have argued that the trade-off between robustness and accuracy might be unavoidable. However, their setting involves a distribution that is not robustly separable by any classifier. In such a situation there is indeed a trade-off between robustness and accuracy. In this section, we focus on settings where robust classifiers exist, which is a more realistic scenario for real-world data. At least for vision, one may well argue that "humans" are robust classifiers, and as a result, we would expect that classes are well-separated at least in some representation space. In fact, Yang et al. [280] show that classes are already well-separated in the input space. In such situations, there is no need for robustness to be at odds with accuracy. A more plausible scenario which we posit, and provide theoretical evidence in support of in Theorem 4, is that depending on the choice of representations, the trade-off may exist or can be avoided. Recent empirical work [173] including our work in Chapter 6 has also established that modifying the training objective to favour certain inductive bias in the learned representations can automatically lead to improved robustness.

On a related note, it has been suggested in recent works that adversarially robust learning may require more "complex" decision boundaries, and as a result may require more data [238, 234, 285, 172]. However, the question of decision boundaries in neural networks is subtle as the network learns a *feature representation* as well as a decision boundary on top of it. We develop concrete theoretical examples in Theorems 3 and 4 to establish that choosing one feature representation over another may lead to *visually* more complex decision boundaries on the input space, though these



(a) Both Parity and Union of Interval classifier predict red if inside any green interval and blue if outside all intervals. The ×-es are correctly labelled and the ∘-es are mislabeled points. Reference integer points on the line labelled in *binary*.

(b) Robust generalisation needs more complex boundaries

Figure 5.6: Visualisation of the distribution and classifiers used in the Proof of Theorems 3 and 4. The Red and Blue indicate the two classes.

are not necessarily more complex in terms of statistical learning theoretic concepts such as VC dimension.

5.3.1 Representation learning without label noise

The choice of inductive biases incorporated in a model affects representations and introduces desirable and possibly even undesirable (cf. [167]) invariances; for example, training convolutional networks are invariant to (some) translations, while training fully connected networks are invariant to permutations of input features. This means that fully connected networks can learn even if the pixels of each training image in the training set are permuted with a fixed permutation [292]. This invariance is worrying as it means that such a network can effectively classify a matrix (or tensor) that is visually nothing like a real image into an image category.

In this section, we present a result to show that there exists a data distribution where proper representation is necessary for small adversarial error as well as small test error whereas another representation can provide low test error but necessarily have large adversarial error. Interestingly, the representation that can achieve small adversarial error can look visually more complex due to the larger number of distinct linear regions in its decision boundary. However, statistically, it will have a smaller VC dimension than its counterpart. We first present the theorems with a proof sketch for ease of understanding and the more detailed proofs in Appendix D.1.

Theorem 3 (Representation Learning for Adversarial Robustness)**.** *For some universal constant c, and any $0 < \gamma_0 < 1/\sqrt{2}$, there exists a family of distributions $\mathcal{D}$ defined on $\mathcal{X} \times \{0, 1\}$ where $\mathcal{X} \subseteq \mathbb{R}^2$ such that for all distributions $\mathcal{P} \in \mathcal{D}$, and*

denoting by $\mathcal{S}_m = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_m, y_m)\}$ *a sample of size* m *drawn i.i.d. from* $\mathcal{P}$,

(i) For any $m \geq 0$, $\mathcal{S}_m$ *is linearly separable i.e.,* $\forall (\mathbf{x}_i, y_i) \in \mathcal{S}_m$, *there exist* $\mathbf{w} \in \mathbb{R}^2, w_0 \in \mathbb{R}$ *s.t.* $y_i \left(\mathbf{w}^\top \mathbf{x}_i + w_0\right) \geq 0$. *Furthermore, for every* $\gamma > \gamma_0$, *any linear separator* f *that perfectly fits the training data* $\mathcal{S}_m$ *has* $\mathcal{R}_{\mathrm{Adv},\gamma}(f; \mathcal{P}) \geq 0.0005$, *even though* $\mathcal{R}(f; \mathcal{P}) \to 0$ *as* $m \to \infty$.

(ii) There exists a function class $\mathcal{H}$ *such that for some* $m \in O(\log(\delta^{-1}))$, *any* $h \in \mathcal{H}$ *that perfectly fits the* $\mathcal{S}_m$, *satisfies with probability at least* $1 - \delta$, $\mathcal{R}(h; \mathcal{P}) = 0$ *and* $\mathcal{R}_{\mathrm{Adv},\gamma}(h; \mathcal{P}) = 0$, *for any* $\gamma \in [0, \gamma_0 + 1/8]$.

A complete proof of this result appears in Appendix D.1, but first, we provide a sketch of the key idea here. The distributions in family $\mathcal{D}$ will be supported on balls of radius at most $1/\sqrt{2}$ on the integer lattice in $\mathbb{R}^2$. The *true* class label for any point $\mathbf{x}$ is provided by the parity of $a + b$, where (a, b) is the lattice point closest to $\mathbf{x}$. However, the distributions in $\mathcal{D}$ are chosen to be such that there is also a linear classifier that can separate these classes, e.g. a distribution only supported on balls centred at the points (a, a) and $(a, a + 1)$ for some integer a (See Figure 5.6(b)). *Visually* learning the classification problem using the parity of $a + b$ results in a seemingly more complex decision boundary, a point that has been made earlier regarding the need for more complex boundaries to achieve adversarial robustness [64, 238]. However, it is worth noting that this complexity is not rooted in any *statistical theory*, e.g. the VC dimension of the classes considered in Theorem 3 is essentially the same (even lower for $\mathcal{H}$ by 1). This *visual* complexity arises purely because the linear classifier looks at a geometric representation of the data whereas the parity classifier looks at the binary representation of the sum of the nearest integer of the coordinates. In the case of neural networks, recent works [136] have indeed provided empirical results to support that excessive invariance (eg. rotation invariance) increases adversarial error.

5.3.2 Representation learning with label noise

In this section, we show how the choice of representation is important in the presence of label noise to learn an adversarially robust classifier. Informally, we show that

if the *correct* representation is used, then in the presence of label noise, it will be impossible to fit the training data perfectly, but the classifier that best fits the training data will have good test accuracy and adversarial accuracy. However, using an "incorrect" representation, we show that it is possible to find a classifier that has zero training error, has good test accuracy, but has a high *adversarial error*. We posit this as a (partial) explanation of why classifiers trained on real data (with label noise) have good test accuracy, while still being vulnerable to adversarial attacks.

Theorem 4 (Representation Learning in the Presence of Noise)**.** *For any $n \in \mathbb{Z}_+$, there exists a family of distributions $\mathcal{D}^n$ over $\mathbb{R} \times \{0,1\}$ and function classes $\mathcal{C}, \mathcal{H}$, such that for any $\mathcal{P}$ from $\mathcal{D}^n$, and for any $0 < \gamma < 1/4$, and $\eta \in (0, 1/2)$ if $\mathcal{S}_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ denotes a sample of size m drawn from $\mathcal{P}$ where*

$$m = \mathcal{O}\left(\max\left\{n \log \frac{n}{\delta}\left(\frac{(1-\eta)}{(1-2\eta)^2} + 1\right), \frac{n}{\eta\gamma^2}\log\left(\frac{n}{\gamma\delta}\right)\right\}\right)$$

and if $\mathcal{S}_{m,\eta}$ denotes the sample where each label is flipped independently with probability η.

(i) the classifier $c \in \mathcal{C}$ that minimises the training error on $\mathcal{S}_{m,\eta}$, has $\mathcal{R}(c; \mathcal{P}) = 0$ and $\mathcal{R}_{\mathrm{Adv},\gamma}(c; \mathcal{P}) = 0$ for $0 \leq \gamma < 1/4$.

(ii) there exist $h \in \mathcal{H}$, h has zero training error on $\mathcal{S}_{m,\eta}$, and $\mathcal{R}(h; \mathcal{P}) = 0$. However, for any $\gamma > 0$, and any $h \in \mathcal{H}$ with zero training error on $\mathcal{S}_{m,\eta}$, $\mathcal{R}_{\mathrm{Adv},\gamma}(h; \mathcal{P}) \geq 0.1$.

Furthermore, the required $c, h \in \mathcal{C}, \mathcal{H}$ can be computed in $\mathcal{O}\left(\mathrm{poly}\,(n), \mathrm{poly}\left(\frac{1}{\frac{1}{2}-\eta}\right), \mathrm{poly}\left(\frac{1}{\delta}\right)\right)$.

We sketch the proof here and present the complete proof in Appendix D.1; as in Theorem 3, we will make use of parity functions, though the key point is the representations used. Let $\mathcal{X} = [0, N]$, where $N = 2^n$, we consider distributions that are supported on intervals $(i - 1/4, i + 1/4)$ for $i \in \{1, \ldots, N-1\}$ (See Figure 5.6(a)), but any such distribution will only have a small number, $O(n)$, intervals on which it is supported. The *true* class label is given by a function that depends on the parity of some hidden subsets S of bits in the bit-representation of the closest integer i, e.g. as in Figure 5.6(a) if $S = \{0, 2\}$, then only the least significant and the third least

significant bit of i are examined and the class label is 1 if an odd number of them are 1 and 0 otherwise. Despite the noise, the *correct* label on any interval can be guessed by using the majority vote and as a result, the correct parity learnt using Gaussian elimination. (This corresponds to class $\mathcal{C}$ in Theorem 4.) On the other hand, it is also possible to learn the function as a union of intervals, i.e. find intervals, $I_1, I_2, \ldots, I_k$ such that any point that lies in one of these intervals is given the label 1 and any other point is given the label 0. By choosing intervals carefully, it is possible to fit *all the training data*, including noisy examples, but yet not compromise on *test accuracy* (Fig. 5.6(a)). Such a classifier, however, will be vulnerable to adversarial examples by applying Theorem 2. A classifier such as a union of intervals ($\mathcal{H}$ in Theorem 4)is translation-invariant, whereas the parity classifier is not. This suggests that using classifiers, such as neural networks, that are designed to have too many built-in invariances might hurt its robustness accuracy. In Section 5.4.1, we present further experimental evidence that neural networks trained with SGD learn more linear-like (simpler) decision boundaries than is necessary for obtaining adversarial robustness.

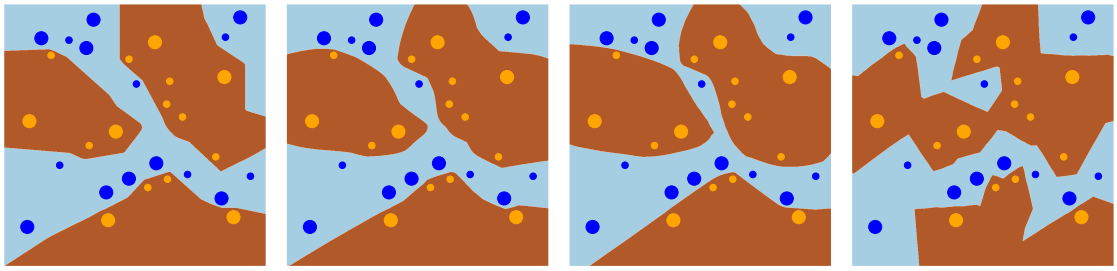
5.4 Experimental results on the impact of representation learning

This section discusses the importance of representation learning for adversarial robustness in the context of neural networks. In particular, we show that neural networks are more linear-like than is required for them to be adversarially robust. Then, we suggest a way to learn richer representations and show that this helps with adversarial robustness.

5.4.1 Complexity of decision boundaries

When neural networks are trained using SGD like algorithms they create classifiers whose decisions boundaries are geometrically much simpler than they need to be for being adversarially robust. A few recent studies [238, 234] have discussed that robustness might require more complex classifiers. In Theorems 3 and 4 we discussed

this theoretically and also why this might not violate the traditional wisdom of Occam's Razor. In particular, complex decision boundaries does not necessarily mean more complex classifiers in statistical notions of complexity like the VC dimension. In this section, we show through a simple experiment how the decision boundaries of neural networks are not "complex" enough to provide large enough margins and are thus adversarially much more vulnerable than is possible.

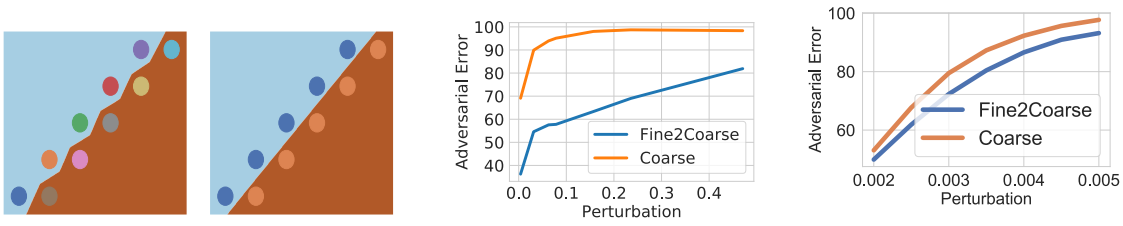


(a) Shallow NN (b) Shallow-Wide NN (c) Deep NN (d) Large Margin

Figure 5.7: Decision boundaries of neural networks are much simpler than they should be.

We train three different neural networks with ReLU activations, a shallow network (Shallow NN) with 2 layers and 100 neurons in each layer, a shallow network with 2 layers and 1000 neurons in each layer (Shallow-Wide NN), and a deep network with 4 layers and 100 neurons in each layer. We train them for 200 epochs on a binary classification problem as constructed in Figure 5.7. The distribution is supported on blobs and the colour of each blob represents its label. On the right side, we have the decision boundary of a large margin classifier, which is represented by a 1-nearest neighbour algorithm.

From Figure 5.7, it is evident that the decision boundaries of neural networks trained with standard optimisers have far *simpler* decision boundaries than is needed to be robust (eg. the 1- nearest neighbour is much more robust than the neural networks.). In particular, the distinction between neural networks and the large margin classifier can be noticed clearly in the decision boundary between the blue and the orange balls in the top left part of the images in Figure 5.7. In an effort to have a less jagged decision boundary for neural networks, the boundary passes very close to the data (the blue and orange balls) for the neural networks than it does for the large margin classifier. This bias towards simplicity for NNs trained using SGD, we hypothesise, is partly responsible for the increased vulnerability of neural



(a) MULTI-CLASS (b) NATURAL (c) CIFAR-100. (d) Restricted Imagenet

Figure 5.8: Assigning a separate class to each sub-population within the original class during training increases robustness by learning more meaningful representations.

networks.

5.4.2 Accounting for sub-populations leads to better robustness

One way to evaluate whether more meaningful representations lead to better robust accuracy is to use training data with more fine-grained labels (e.g. subclasses of a class); for example, one would expect that if different breeds of dogs are labelled differently the network will learn features that are relevant to that extra information. We show using synthetic data, CIFAR100 [145], and Restricted Imagenet [259] that training on fine-grained labels does increase robust accuracy.

We hypothesise that learning more meaningful representations by accounting for fine-grained sub-populations within each class may lead to better robustness. We use the theoretical setup presented in Figure 5.6(b) to conduct our synthetic data experiment.

Recall that the data distributions for the binary learning problem in Figure 5.6(b) is supported on balls of radius at most $1/\sqrt{2}$ on the integer lattice in $\mathbb{R}^2$. The *true* class label for any point $\mathbf{x}$ is provided by the parity of $a + b$, where (a, b) is the lattice point closest to $\mathbf{x}$. As we noted in the proof of Theorem 3, the distribution is separable by a *simple* linear classifier by a small margin. However, if each of the circles belonged to a separate class then the decision boundary would have to be necessarily more complex as it needs to, now, separate the balls that were previously within the same class. We test this hypothesis with two experiments. First, we test it on the distribution defined in Theorem 3 where for each ball with label 1, we assign

it a different label (say $\alpha_1, \ldots, \alpha_k$) and similarly for balls with label 0, we assign it a different label ($\beta_1, \ldots, \beta_k$). Now, we solve a multi-class classification problem for $2k$ classes with a deep neural network and then later aggregate the results by reporting all α_is as 1 and all β_is as 0. The resulting decision boundary is drawn in Figure 5.8(a) along with the decision boundary for natural training in Figure 5.8(b). Clearly, the margin for the multi-class model (and thus robustness) is greater than the naturally trained model.

Second, we also repeat the experiment with CIFAR-100 and Restricted Imagenet [259] in Figure 5.8(c) and Figure 5.8(d) respectively. (see Appendix B for details on the datasets). For CIFAR-100, we train a ResNet50 [110] on the fine labels of CIFAR100 and then aggregate the fine labels corresponding to a coarse label by summing up the logits of the fine classes corresponding to each coarse class. For restricted imagenet, we use the fine-coarse division mentioned in Table B.1. We call this model the *Fine2Coarse* model and compare the adversarial risk of this network to a ResNet-50 trained directly on the coarse labels. Note that the model is end-to-end differentiable as the only addition is a layer to aggregate the logits corresponding to the fine classes of each coarse class. Thus PGD adversarial attacks can be applied off the shelf. Figures 5.8(c) and 5.8(d) show that for all perturbation budgets, *Fine2Coarse* has a smaller adversarial risk than the naturally trained model.

5.4.3 Discussion and other relevant works

A related result by Montasser et al. [185] shows that certain hypothesis classes are only *improperly* robustly PAC learnable despite having a finite VC dimension. Thus, learning the problem with small adversarial error requires using a different class of models (or representations) whereas, for small natural test risk, the original model class (or representation) can be used (it is properly learnable due to its finite VC dimension). In particular, the examples from Montasser et al. [185] that uses improper learning to learn a robust classifier has a much higher sample complexity. In our example, learning algorithms for both the hypotheses classes that we use have polynomial sample complexity. Another point of distinction is that Theorem 4

uses a training set induced with random classification noise and hypothesis class $\mathcal{H}$ obtains zero training on this noisy training set whereas the examples in Montasser et al. [185] do not have any label noise.

Hanin and Rolnick [105] have shown that though the number of possible linear regions that can be created by a deep ReLU network is exponential in depth, in practice for networks trained with SGD this tends to grow only linearly thus creating much simpler decision boundaries than is possible due to sheer expressivity of deep networks. Experiments on the data models from our theoretical settings show that adversarial training indeed produces more "complex" decision boundaries

Jacobsen et al. [127] have discussed that excessive invariance in neural networks might increase adversarial error. However, they argue that excessive invariance can allow sufficient changes in the semantically important features without changing the network's prediction. They describe this as Invariance based adversarial examples as opposed to perturbation based adversarial examples. We show that excessive (incorrect) invariance might also result in perturbation based adversarial examples.

Another contemporary work [83] discusses a phenomenon they refer to as *Shortcut Learning* where deep learning models perform very well on standard tasks like reducing classification error but fail to perform in more difficult real-world situations. We discuss this in the context of models that have small test error but large adversarial error and provide theoretical and empirical to discuss why one of the reasons for this is sub-optimal representation learning.

6. Improving Adversarial Robustness via low-rank representations

Dimensionality reduction methods are some of the oldest techniques in machine learning that extract a small number of factors from a dataset that explains almost all of its variance; these factors contain most of the *discriminative* power useful in classification or regression tasks and are known to increase *robustness*, i.e. these methods typically have a denoising effect. Perhaps, the most popular and widely used among them are PCA (see e.g. [134]) and CCA [118].

An intriguing aspect of deep neural networks has been their ability to learn representations directly from the raw data that are useful in several tasks, including ones for which they were not specifically trained, usually known as *representation learning* [290, 237, 70, 146, 96, 109, 266, 217]. Essentially, for most models trained in a supervised fashion, the vector of activations in the penultimate layer is a *learned* representation of the raw input. The remarkable success of Deep Neural Networks (DNNs) is primarily attributed to the discriminative quality of this representation space. However, despite their impressive performance, DNNs are known to be brittle to input perturbations as we discussed in the previous chapter [248, 93, 62, 29, 43, 205, 186]. In this chapter, we study the importance of proper inductive biases in the representation space for adversarial robustness. In particular, we look at whether reducing the intrinsic dimensionality of the representation space helps with adversarial robustness.

6.1 Dimensionality of representations and adversarial robustness

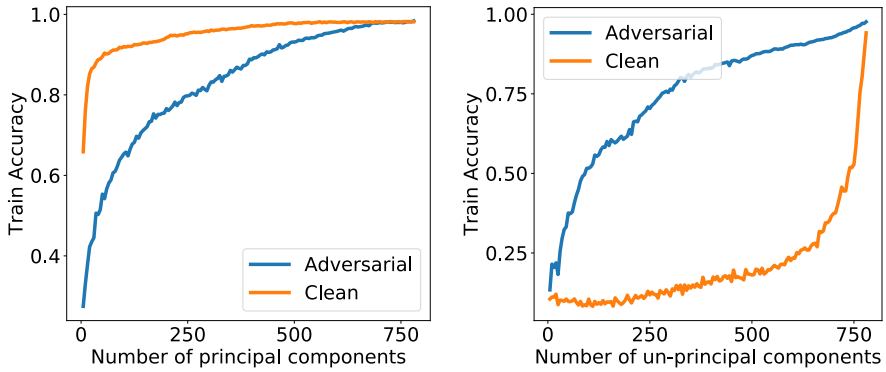
The vulnerability of deep neural networks towards adversarial perturbations raises concerns regarding the robustness of the factors captured by these learned representations. On the other hand, as mentioned earlier, dimensionality reduction techniques capture factors that are, while being discriminative, robust to input per-

turbations. This motivates the thesis behind this chapter—*if we enforce DNNs to learn representations that lie in a low-dimensional subspace (for the entire dataset), we might be able to obtain more robust classifiers while preserving their discriminative power.*

Principal and un-principal components of representation space To get further insights into why restricting the dimensionality of representation space is helpful for adversarial robustness, we perform a simple experiment that indicates that adversarial attacks exploit "un-principal" components, i.e. components corresponding to the smallest eigenvalues of the covariance matrix of the data. We first train a six-layer neural network with four convolutional and two fully connected layers on the MNIST dataset to convergence and attack it with a PGD adversary (see Section 3.2) to obtain adversarial examples. Then, we project the adversarial examples on the *top* k and the *bottom* k PCA components of the full MNIST dataset, for all $1 \leq k \leq 784$, to obtain k sets each of projections of the adversarial examples on the top and bottom k components, respectively. We will refer to these $2k$ sets of projections as the *principal* and *un-principal* components of the adversarial examples, respectively. Then for varying k, we train k separate linear classifiers to predict the adversarial labels using just the k principal components of the adversarial examples and measure its training accuracy. Similarly, we train k separate classifiers to predict the adversarial label[1] using just the k un-principal components. We plot these training accuracies in blue in Figure 6.1 (starting from the principal k components in the left figure, and un-principal k components in the right figure). Training accuracy for the k components is a measure of how *easy* it is to fit the data with just those k components and is thus, a measure of the amount of discriminatory information contained in them. A similar procedure is done for a randomly sub-sampled set (of the same size as the number of adversarial examples) of (clean) training images and their training accuracy on the clean labels is plotted in orange.

Figure 6.1 shows that for the k principal components, the training accuracy

[1]An adversarial label is the incorrect label the model predicts for the adversarial example.



Top k (Larger Variance) Bottom k (Lesser variance)

Figure 6.1: Adversarial Noise manipulates the principal components with lesser variance.

increases much faster on the clean images for a relatively small number of principal components. For the k un-principal components, the training accuracy increases faster for the adversarial examples. For *clean examples*, almost all the *discriminatory information* lies in the principal components. For adversarial examples, a significant amount of the discriminatory information for predicting the adversarial labels are contained in the un-principal components. This suggests that we should find low-dimensional representations that retain the discriminatory information needed for high test accuracy while removing "noisy" components that could be exploited by an adversary. While for simple data, PCA can be used to do this, when using deep neural networks, a method to induce low-rankness in some hidden layers (learned representations) is needed, which is what this chapter provides.

Challenges to obtain low-dimensional representations Ideally, to encourage learning low-dimensional representations, we would like to insert a *dimensionality reduction* "module" in deep neural networks and develop an end-to-end training method that simultaneously does supervised training and dimensionality reduction. At first glance, using SVD to project representations onto a low-dimensional subspace seems viable. However, this approach encounters multiple challenges.

1. The SVD algorithm needs to operate on a matrix one of whose dimensions is as large as the number of examples in the dataset. As the number of examples is usually very large, executing the SVD algorithm can be computationally prohibitive in practice.

2. The representations themselves are not learnable parameters but just parametric functions of the input data; as a consequence, they change after every parameter update. It is not straightforward how to preserve the low rank of the representations, after doing the SVD, from one iteration to the next.

A workaround could be to design architectures with bottlenecks similar to auto-encoders [115]. The fact that most of the state-of-the-art networks do not have such bottlenecks limit their usability. Further, as we discuss in Section 6.5, it also doesn't improve adversarial robustness in practice. We discuss this and other alternate algorithms in Section 6.5.

6.2 Main contributions

This algorithm proposed in this chapter provides the benefits of dimensionality reduction by inserting a *virtual layer* (not used at prediction time) and augmenting the loss function to induce low-rank representations.

Algorithmic Contributions We propose a novel low-rank regulariser (LR) to control the intrinsic dimensionality of the representations that

1. does not put any restriction on the network architecture,[2]
2. is end-to-end trainable, and
3. is efficient in that it allows mini-batch training.

LR explicitly *enforces* representations to lie in a linear subspace with low *intrinsic* dimension and is guaranteed to provide low-rank representations for the entire dataset even when trained using mini-batches. As LR is a virtual layer (discussed in later sections), it can be applied to any intermediate representations of DNNs.

Experimental Contributions Experimentally, apart from successfully reducing the dimensionality of learned representations, neural networks trained with LR turn out to be significantly more robust to input perturbations, both adversarial and random, while providing modest improvements over the natural (unperturbed) test

[2]It puts no *direct* restriction, though of course, any extra regularisation will produce an inductive bias.

accuracy. This is of particular interest as it provides empirical evidence that adding well-thought priors over factors influencing the representation space (e.g. low-rank prior over representations) might further improve the robustness of DNNs, without encountering a trade-off with, be it computational [93, 172], statistical [234] or a loss in accuracy [259]. See Section 3.2.4 for a discussion on the various tradeoffs associated with this. This is in line with recent works, including Chapter 5 of this thesis, suggesting that a "correct" representation may avoid the perceived trade-off between robustness and accuracy (see [172, 294, 185]).

Lastly, because of the low dimensionality, we can compress representations by a significant factor without losing its discriminative power. Thus, discriminative features of the data can be stored using very little memory. For example, we show in one of our experiments that, even with a 5-dimensional embedding (400x compression), a model with LR loses only 6% in accuracy.

6.3 Deep Low-Rank Representations Layer (LR Layer)

Consider $f : \mathbb{R}^d \mapsto \mathbb{R}^k$ to be a feed-forward MLP that maps d dimensional input $\mathbf{x}$ to a k dimensional output $\mathbf{y}$. We can decompose this into two sub-networks, one consisting of the layers before the ℓ^{th} layer and one after i.e. $f(\mathbf{x}) = f_\ell^+\left(f_\ell^-(\mathbf{x};\phi);\theta\right)$, where $f_\ell^-(.;\phi)$, parameterised by ϕ, represents the part of the network up to layer ℓ and, $f_\ell^+(.;\theta)$ represents the part of the network thereafter. With this notation, the m dimensional representation (or the activations) after any layer ℓ can simply be written as $\mathbf{a} = f_\ell^-(\mathbf{x};\phi) \in \mathbb{R}^m$. In what follows, we first formalise the low-rank representation problem and then propose our approach to solve it approximately and efficiently.

6.3.1 Formulating the LR Layer problem

Let $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ and $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^N$ be the set of inputs and target outputs of a given training dataset. By slight abuse of notation, we define $\mathbf{A}_\ell = f_\ell^-(\mathbf{X};\phi) = [\mathbf{a}_1, \cdots, \mathbf{a}_N]^\top \in \mathbb{R}^{N\times m}$ to be the activation matrix of the entire dataset, so that $\mathbf{a}_i$

is the activation vector of the i-th sample. Note that for most practical purposes $N \gg m$. In this setting, the problem of learning low-rank representations can be formulated as a constrained optimisation problem as follows:

$$\min_{\theta,\phi} \mathcal{L}(\mathbf{X}, \mathbf{Y}; \theta, \phi), \text{ s.t. } \text{rank}\,(\mathbf{A}_\ell) = r, \tag{6.1}$$

where $\mathcal{L}(.)$ is the loss function and $r < m$ is the desired rank of the representations at layer ℓ. The rank r is a hyperparameter (though empirically not a sensitive one as observed in our experiments). Throughout this section, we discuss the problem of imposing low-rank constraints over a single intermediate layer, however, it can trivially be extended to any number of layers. Note that both the loss and the constraint set of the above objective function are non-convex. One approach to optimising this would be to perform an alternate minimisation style algorithm (eg. [169, 65]), first over the loss (gradient descent) and then projecting onto the non-convex set to satisfy the rank constraint.

Since $N \gg m$, ensuring $\text{rank}\,(\mathbf{A}_\ell) = r$ would be practically infeasible as it would require performing SVD in every iteration at a cost $\mathcal{O}(N^2 m)$. A feasible, but incorrect, approach would be to do this on mini-batches, instead of the entire dataset. Projecting each mini-batch onto the space of rank r matrices does not guarantee that the activation matrix of the entire dataset will be of rank r, as each of these mini-batches can lie in very different subspaces. As a simple example, consider the set of coordinate vectors which can be looked at as rank one matrices corresponding to activations in batches of size one. However, when these coordinate vectors are stacked together to form a matrix they create the identity matrix, which is a full rank matrix.

Computational issues aside, another crucial problem stems from the fact that the activation matrix $\mathbf{A}_\ell = f_\ell^-(.; \phi)$ is itself parameterised by ϕ and thus ϕ needs to be updated in a way such that the generated $\mathbf{A}_\ell$ is low rank. It is not immediately clear how to use the low-rank projection of $\mathbf{A}_\ell$ to achieve this. One might suggest to first fully train the network and then obtain low-rank projections of the activations. Our experiments show that this procedure does not provide the two main benefits

we are looking for: preserving accuracy under compression and robustness.

Low-Rank regulariser: We now describe our regulariser that encourages learning low-rank activations, and, if optimised properly, guarantees that the rank of the activation matrix (of any size) will be bounded by r. The primary ingredient of our approach is the introduction of an auxiliary parameter $\mathbf{W} \in \mathbb{R}^{m \times m}$ in a way that allows us to shift the low-rank constraint from the activation matrix $\mathbf{A}_\ell$ (as in Equation (6.1)) to $\mathbf{W}$ providing the following two advantages: The rank constraint is now

1. on a matrix that is independent of the batch/dataset size, and
2. on a parameter as opposed to a data-dependent intermediate tensor (like activations), and can thus be updated directly at each iteration.

Combining these ideas, our final augmented objective function, with the regulariser, is:

$$\begin{aligned} &\min_{\theta,\phi,\mathbf{W},\mathbf{b}} \mathcal{L}(\mathbf{X},\mathbf{Y};\theta,\phi) + \lambda_1 \mathcal{L}_c(\mathbf{A}_\ell;\mathbf{W},\mathbf{b}) + \lambda_2 \mathcal{L}_n(\mathbf{A}_\ell) \\ &\text{s.t.,} \mathbf{W} \in \mathbb{R}^{m \times m}, \operatorname{rank}(\mathbf{W}) = r,\ \mathbf{b} \in \mathbb{R}^m, \mathbf{A}_\ell = f_\ell^-(\mathbf{X};\phi), \end{aligned} \tag{6.2}$$

where,

$$\mathcal{L}_c(\mathbf{A}_\ell;\mathbf{W},\mathbf{b}) = \frac{1}{n}\sum_{i=1}^{n} \left\| \mathbf{W}^\top(\mathbf{a}_\mathrm{i} + \mathbf{b}) - (\mathbf{a}_\mathrm{i} + \mathbf{b}) \right\|_2^2, \text{and } \mathcal{L}_n(\mathbf{A}_\ell) = \frac{1}{n}\sum_{i=1}^{n} \Big| 1 - \|\mathbf{a}_\mathrm{i}\| \Big|.$$

Here, $\mathcal{L}_c$ is the projection loss that ensures that the affine low-rank mappings ($\mathbf{AW}$) of the activations are close to the original ones i.e. $\mathbf{AW} \approx \mathbf{A}$. As the constraint ($\operatorname{rank}(\mathbf{W}) = r$) ensures that $\mathbf{W}$ is low-rank, $\mathbf{AW}$ is also low-rank and thus implicitly (due to $\mathbf{AW} \approx \mathbf{A}$), $\mathcal{L}_c$ forces the original activations $\mathbf{A}$ to be low-rank. The bias $\mathbf{b}$ allows for the activations to be translated before projection.[3]

However, note that setting $\mathbf{A}$ and $\mathbf{b}$ close to zero trivially minimises $\mathcal{L}_c$, especially when the activation dimension is large. Also, due to the positive homogeneity of each

[3]We use the term *projection* loosely as we do not strictly constrain $\mathbf{W}$ to be a projection matrix.

layer, the magnitude of the activations can be minimised in a layer by multiplying the weights of that layer by a small constant c and then maximised in the next layer by multiplying the weights of that layer with $\frac{1}{c}$ thereby preserving the final logit magnitudes. We observed this to happen in practice as it is easier for the network to learn ϕ such that the activations and the bias are very small to minimise $\mathcal{L}_c$ as compared to learning a low-rank representation space. To prevent this, we use $\mathcal{L}_n$ that acts as a norm constraint on the activation vector to keep the activations sufficiently large. Lastly, as the rank constraint is now over $\mathbf{W}$ and $\mathbf{W}$ is a *global* parameter independent of the dimension n (i.e. size of mini-batch/dataset), we can use mini-batches to optimise Equation (6.2). Since $\text{rank}\,(\mathbf{AW}) \leq r$ for any $\mathbf{A}$, optimizing over mini-batches still ensures that the entire activation matrix is low-rank. Intuitively, this is because the basis vectors of the low-rank affine subspace are now captured by the low-rank parameter $\mathbf{W}$. Thus, as long as $\mathcal{L}_c$ is minimised for all mini-batches, $\mathbf{A} \approx \mathbf{AW}$ holds for the entire dataset, leading to the low-dimensional support. Thus, the overall goal of the augmented objective function in Equation (6.2), is to jointly penalise the activations ($\mathbf{A}_\ell$) to make them low-rank and learn the corresponding low-rank identity map $(\mathbf{W}, b)$. Note, implementation wise, our regulariser requires adding a virtual (does not modify the main network) branch with parameters $(\mathbf{W}, b)$ at layer ℓ. This branch is removed at the time of inference as the activations learned, by virtue of our objective function, are already low-rank.

Remark 7. *The reason we need to minimise both the reconstruction loss $\mathcal{L}_c$ and the norm constraint loss $\mathcal{L}_n$ simultaneously is the same as why the spectral norm and the stable rank can be independently minimised in Chapter 4 without affecting each other. To see why, note that the rank and the stable rank of the matrix is independent of the scale of the matrix i.e. the matrix can be multiplied by any non-zero scalar without affecting the rank or the stable rank of the matrix whereas multiplying the matrix by a scalar affects the spectral norm proportionally.*

Hyper-parameters While our algorithm has three hyper-parameters λ_1, λ_2 and r, in practice, our approach is insensitive to λ_1, λ_2 and thus, we set $\lambda_1 = \lambda_2 = 1$. This

Algorithm 6 Low-Rank (LR) regulariser

input Activation Matrix $\mathbf{A}_l$, gradient input $\mathbf{g}_l$

$\mathbf{Z} \leftarrow (\mathbf{A}_l + \mathbf{b})^\top \mathbf{W}$ {forward propagation towards the virtual LR layer}

$\mathcal{L}_c \leftarrow \frac{1}{b}\|\mathbf{Z} - (\mathbf{A}_l + \mathbf{b})\|_2^2$ {the reconstruction loss}

$\mathcal{L}_n \leftarrow \frac{1}{b}\sum_{i=1}^{b} \left|\mathbf{1} - \|\mathbf{a}_i\|\right|$ {norm constraint loss}

$\mathbf{g}_W \leftarrow \frac{\partial \mathcal{L}_c}{\partial \mathbf{W}}$, $\mathbf{g} \leftarrow \mathbf{g}_l + \frac{1}{b}\sum_{i=1}^{b} \frac{\partial(\mathcal{L}_c + \mathcal{L}_n)}{\partial \mathbf{a}_i}$

$\mathbf{W} \leftarrow \mathbf{W} - \lambda \mathbf{g}_W$

$\mathbf{W} \leftarrow \Pi_k^{\text{rank}}(\mathbf{W})$ {hard thresholds the rank of $\mathbf{W}$}

output $\mathbf{g}$ {the gradient to be passed to the layer before}

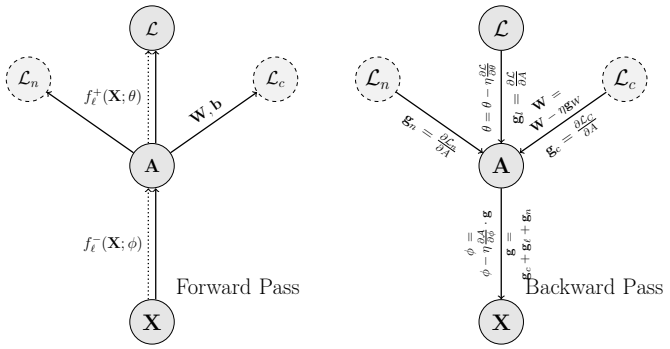
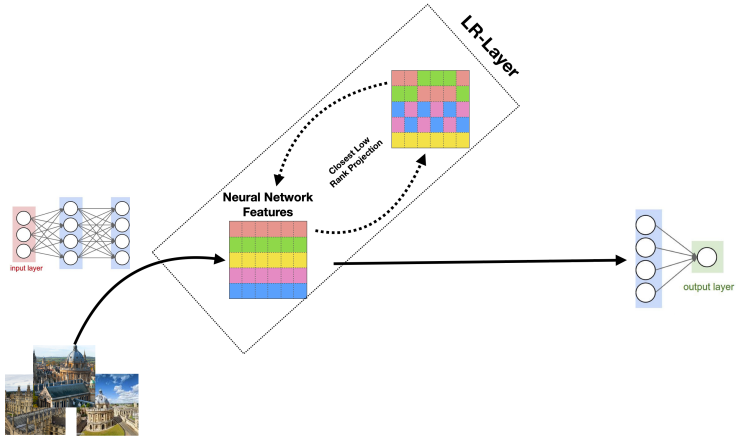


Figure 6.2: **The LR layer**. The left figure shows the *forward pass*, *solid edges* show the flow of data during training, *dashed edges*- the flow of data during inference, and *dashed nodes* are the *virtual layer*. The right figure shows the *backward pass*.

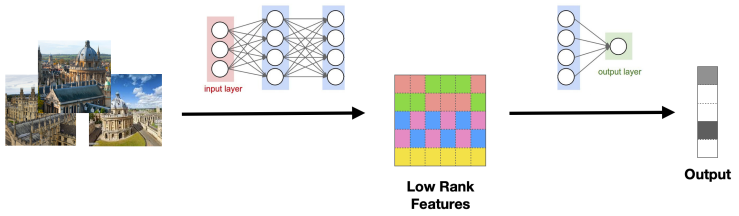
is an added advantage given the resources needed to do hyper-parameter optimisation. In the classical view of regularisation e.g. ridge regression, the regularisation coefficient induces a bias-variance trade-off i.e. as $\|\mathrm{W}\|_\mathrm{F} \to 0$, the accuracy decreases. In our case, under the assumption that there exist low-rank representations achieving zero classification error, even as the terms $\mathcal{L}_n$ and $\mathcal{L}_c$ go to 0, the original classification loss $\mathcal{L}$ does not (necessarily) increase. One way of thinking about this is that the terms $\mathcal{L}_n$ and $\mathcal{L}_c$ are guiding the optimisation to specific minimisers of the empirical classification loss, of which there will necessarily be several because of overparameterisation. To verify this empirically, we ran experiments with all combinations of λ_1, λ_2 chosen from $\{1., 0.1, 0.01\}$. We observe that for all these settings, the terms $\mathcal{L}_n$ and $\mathcal{L}_c$ go to 0 while the original loss does not degrade at all.

6.3.2 Algorithm for LR Layer

We solve our optimisation problem (Equation (6.2)) by adding a *virtual* low-rank layer that penalises representations that are far from their closest low-rank affine representation. Algorithm 6 describes the operation of the low-rank virtual layer for



(a) Training



(b) Testing

Figure 6.3: Figure 6.3(a) shows an illustration of how the LR-Layer is added as a plug-and-play layer during training. Figure 6.3(b) shows an illustration that the LR layer is removed during testing and the generated representations are already low-rank.

a mini-batch of size b. We present a flow diagram for the same in Figure 6.2. This layer is virtual in the sense that it only includes the parameters $\mathbf{W}$ and $\mathbf{b}$ that are not used in the NN model itself to make predictions, but nonetheless, the corresponding loss term $\mathcal{L}_c$ does affect the model parameters through gradient updates. Figure 6.3 provides an illustration of how the LR-Layer is added during training as a plug-and-play layer and then removed during testing, and the generated features are already low rank. Algorithm 6 alternately minimises the augmented loss function in Equation (6.2) and projects the auxiliary parameter $\mathbf{W}$ to the space of low-rank matrices.

The rank projection step in Algorithm 6 is executed by a hard thresholding operator $\Pi_r^{\text{rank}}(W)$, which finds the best r-rank approximation of W. Essentially, $\Pi_r^{\text{rank}}(W)$ solves the following optimisation problem, which can be solved using singular value decomposition (SVD).

$$\Pi_r^{\text{rank}}(W) = \underset{\text{rank}(Z)=r}{\mathbf{argmin}} \|\mathrm{W} - \mathrm{Z}\|_F^2 \tag{6.3}$$

However, the projection can be very expensive due to the large dimension of the representations space (e.g. 16000). To get around this, we use the ensembled Nyström SVD algorithm [273, 103, 150].

Handling large activation matrices : Singular Value Projection (SVP) introduced in Jain et al. [129] is an algorithm for rank minimisation under affine constraints. In each iteration, the algorithm performs gradient descent on the affine constraints alternated with a rank-k projection of the parameters and it provides recovery guarantees under weak isometry conditions. However, the algorithm has a complexity of $O(mnr)$ where m, n are the dimensions of the matrix and r is the desired low rank. Faster methods for SVD of sparse matrices are not applicable as the matrices in our case are not necessarily sparse. We use the ensembled Nyström method [273, 103, 150] to boost our computational speed at the cost of the accuracy of the low-rank approximation. It is essentially a sampling-based low-rank approximation to a matrix. The algorithm is described in detail in Section 6.3.3. Though the overall complexity for projecting W remains $O(m^2 r)$, the complexity of the hard-to-parallelise SVD step is now $O(r^3)$, while the rest is due to matrix multiplication, which is fast on modern GPUs.

The theoretical guarantees of the Nyström method hold only when the weight matrix of the LR-layer is symmetric and positive semi-definite (SPSD) before each $\Pi_r^{\text{rank}}(\cdot)$ operation. Note that the PSD constraint is actually not a restriction as all projection matrices are PSD by definition. For example, on the subspace spanned by columns of a matrix $\mathbf{X}$, the projection matrix is $\mathbf{P} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top$, which is always PSD. We know that a symmetric diagonally dominant real matrix with non-negative diagonal entries is PSD. With this motivation, the matrix $\mathbf{W}$ is smoothened by repeatedly adding $0.01\mathbf{I}$ until the SVD algorithm converges where $\mathbf{I}$ is the identity matrix.[4]. This is a heuristic to make the matrix well-conditioned as well as diagonally dominant and it helps in the convergence of the algorithm empirically.

Symmetric Low-Rank Layer The symmetricity constraint restricts the projections allowed in our optimisation, but empirically this restriction does not seem to

[4]The computation of the singular value decomposition sometimes fail to converge if the matrix is ill-conditioned

hurt its performance. Implementation wise, gradient updates may make the matrix parameter asymmetric, even if we start with a symmetric matrix. Reparameterising the LR-layer fixes this issue; the layer is parameterised using $\mathbf{W}_s$ (to which gradient updates are applied), but the layer projects using $\mathbf{W} = (\mathbf{W}_s + \mathbf{W}_s^\top)/2$, which is by definition a symmetric matrix. After the rank projection is applied to the (smoothed version of) $\mathbf{W}$, $\mathbf{W}_s$ is set to be $\Pi_r^{\text{rank}}(\mathbf{W})$. By Lemma 3, if we start with an SPSD matrix $\mathbf{W}_s$, the updated $\mathbf{W}_s$ is an SPSD matrix. As a result, the updated $\mathbf{W}$ is also SPSD.

Lemma 3 (Column Sampled Nyström approximation preserves SPSD matrices)**.** *If $\mathbf{X} \in \mathbb{R}^{m\times m}$ is an SPSD matrix and $\mathbf{X}_r \in \mathbb{R}^{m\times m}$ is the best Nyström ensembled, column sampled r-rank approximation of $\mathbf{X}$, then $\mathbf{X}_r$ is SPSD as well.*

Proof in Appendix E.1

6.3.3 Ensembled Nyström method

Let $\mathbf{W} \in \mathbb{R}^{m\times m}$ be a symmetric positive semidefinite matrix (SPSD). We want to generate a matrix $\mathbf{W}_r$ which is an r-rank approximation of $\mathbf{W}$ without performing SVD on the full matrix $\mathbf{W}$ but only on a principal submatrix of $\mathbf{W}$. A principal submatrix of a matrix $\mathbf{W}$ is a square matrix formed by removing some columns and the corresponding rows from $\mathbf{W}$ [177]. Let the principal submatrix be $\mathbf{Z} \in \mathbb{R}^{l\times l}$, where $l \ll m$. We construct $\mathbf{Z}$ by first sampling l indices from the set $\{1 \dots m\}$ and selecting the corresponding columns from $\mathbf{W}$ to form a matrix $\mathbf{C} \in \mathbb{R}^{m\times l}$. Then, we select the l rows with the same indices from $\mathbf{C}$ to get $\mathbf{Z} \in \mathbb{R}^{l\times l}$. We can rearrange the columns of $\mathbf{W}$ and $\mathbf{C}$ so that

$$W = \begin{bmatrix} \mathbf{Z} & \mathbf{W}_{21}^T \\ \mathbf{W}_{21} & \mathbf{W}_{22} \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{Z} \\ \mathbf{W}_{21} \end{bmatrix}$$

According to the Nyström approximation, the low-rank approximation of $\mathbf{W}$ can be written as

$$\mathbf{W}_r = \mathbf{C}\mathbf{Z}_r^+\mathbf{C}^T \tag{6.4}$$

where $\mathbf{Z}_r^+$ is the pseudo-inverse of the best r rank approximation of $\mathbf{Z}$. Hence, the entire algorithm is as follows.

- Compute $\mathbf{C}$ and $\mathbf{Z}$ as stated above.
- Compute the top r singular vectors and values of $\mathbf{Z}$: $\mathbf{U}_r,\ \mathbf{\Sigma}_r,\ \mathbf{V}_r$.
- Invert each element of $\mathbf{\Sigma}_r$ to get the Moore pseudo-inverse of $\mathbf{Z}_r$.
- Compute $\mathbf{Z}_r^+ = \mathbf{U}_r\mathbf{\Sigma}_r^{-1}\mathbf{V}_r$ and $\mathbf{W}_r = \mathbf{C}\mathbf{Z}_r^+\mathbf{C}^T$.

Though by trivial computation, the complexity of the algorithm seems to be $O(l^2r + ml^2 + m^2l) = O(m^2r)$ (in our experiments $l = 2r$), however the complexity of the SVD step is only $O(r^3)$ which is much lesser than $O(m^2r)$ and while matrix multiplication is easily parallelisable, SVD is not.

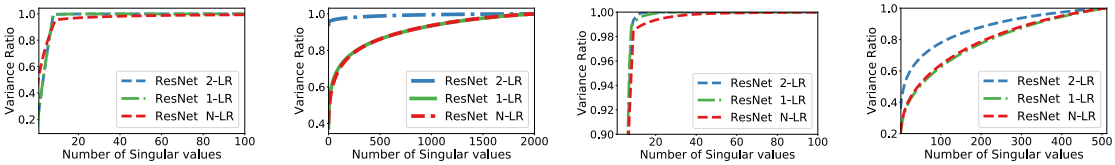
To improve the accuracy of the approximation, we use the ensembled Nyström sampling-based methods [150] by averaging the outputs of t runs of the Nyström method. The l indices for selecting columns and rows are sampled from a uniform distribution and it has been shown [151] that uniform sampling performs better than most other sampling methods. Theorem 3 in Kumar et al. [150] provides a probabilistic bound on the Frobenius norm of the difference between the exact best r-rank approximation of $\mathbf{W}$ and the Nyström sampled r-rank approximation.

6.4 Experiments

We perform a wide range of experiments to show the effectiveness of imposing low-rank constraints on the representations of a dataset using our proposed LR.

Architectures and datasets We use the standard ResNet [110] architecture with four residual blocks. To capture the effect of network depth, we use ResNet-50 (R50) and ResNet-18 (R18). Please refer to Appendix B for more details on datasets and neural network architectures used for the experiments. Since LR can be applied to any representation layer in the network, we investigate the following configurations:

- **1-LR**, where the LR layer is located just before the last fully-connected (FC) layer that contains 512 and 2048 units in ResNet-18 and ResNet-50, respectively.



(a) After last ResNet block. (b) Before last ResNet block (c) After last ResNet block (d) Before last ResNet block

Figure 6.4: Variance Ratio captured by a varying number of Singular Values in ResNet18 trained on CIFAR10 (Figures 6.4(a) and 6.4(b)) and SVHN (Figures 6.4(c) and 6.4(d)). Captions indicate where the activations were extracted from.

- **2-LR**, where there are two LR layers, the first one positioned before the fourth ResNet block with $16,384$ incoming units, and the second one just before the FC layer as in ResNet 1-LR.
- **N-LR**, without any LR layer. This is the standard model without any modification.

We discuss low-rank weight matrices and nuclear norm penalisation as alternate regularisations to reduce the rank of representations in Section 6.5. We use SVHN, CIFAR10 and CIFAR100 datasets and show results using both the coarse labels (20 classes) and fine labels (100 classes) of CIFAR100. Experimentally we observe that the target rank is not a sensitive hyper-parameter, as the training enforces a much lower rank than what it is set to. For our experiments, we set a target rank of 100 for the layer before the last FC layer and 500 for the layer before the fourth ResNet block. The hyper-parameter l in the Nyström method is set to double the target rank. The model is pre-trained with SGD for the first 50 epochs and then Algorithm 6 is applied. The rank cutting operation is performed every 10 iterations.

6.4.1 Effective rank of learned representations

Before we discuss our primary findings, first we empirically show the effect of LR on the effective rank of activations. We use the standard *variance ratio*, defined as $\sum_{i=1}^{r} \sigma_i^2 / \sum_{i=1}^{p} \sigma_i^2$, where σ_i's are the ordered singular values of the given activation matrix $\mathbf{A}$, p is the rank of the matrix, and $r \leq p$. Given r, a higher value of variance ratio indicates that a larger fraction of the total variance in the data is captured in the r dimensional subspace.

Effective Rank Figure 6.4(a) shows the variance ratio for the activations before the last FC layer in ResNet18. Note that even for NLR, the effective rank is as low as 10. A similar low-rank structure was also observed empirically by Oyallon [203]. However, the LR-models have almost negligible variance leakage. Variance leakage for a certain r is defined as $1-$(variance ratio at k).

Figure 6.4(b) shows the variance ratio for the activations before the $4th$ ResNet block. The activation vector is $16,384$-dimensional and the use of the Nyström method ensures computational feasibility. ResNet 2-LR is the only model that has an LR-layer in that position and these figures show that *2-LR is the only model that shows a (reasonably) low-rank structure on that layer.*

In Figures 6.4(c) and 6.4(d), we plot the variance ratio of representations obtained before and after the last resnet block of ResNet18 models trained on SVHN. The LR models show a better low-rank structure than N-LR models and this is consistent with our experiments on CIFAR10.

6.4.2 Adversarial robustness

We now begin our analysis on the impact of low-rank representations on adversarial robustness. We would like to highlight that in all our experiments, all the models are trained using a clean dataset. This is important as it shows whether training on a clean dataset, as opposed to adversarial training [172], with well-thought priors or regularisers, can improve adversarial robustness without compromising on the clean test data accuracy.

We recall that adversarial noise are well crafted (almost imperceptible) input perturbations that, when added to a clean input, flips the prediction of the model to an incorrect one [62, 29, 248]. Various methods [248, 93, 153, 186, 43, 205] have been proposed in recent years for constructing adversarial perturbations. Here we use the following three **white-box** adversarial attacks to perform our experiments: (i) Iterative Fast Sign Gradient Method (Iter-FGSM or IFGSM) [152, 172], (ii) Iterative Least Likely Class Method (Iter-LL-FGSM or ILL) [153], and (iii) DeepFool (DFL) [186]. We discussed these attacks in detail in Section 3.2.2. Iter-FGSM is essentially equivalent to the Projected Gradient Descent (PGD) with ℓ_∞ projections

				Adversarial Test Accuracy(%)								Test Accuracy (%)
L_∞ radius				8/255		10/255		16/255		20/255		
Attack iterations				7	20	7	20	7	20	7	20	
White Box	C10	R50	N-LR	43.1	31.0	38.5	21.8	31.2	7.8	28.9	4.5	**95.4**
			1-LR	**79.1**	**78.5**	**78.6**	**78.1**	**77.9**	**77.0**	**77.1**	**76.6**	**95.4**
		R18	N-LR	40.9	26.7	35.1	16.6	26.7	4.4	24.3	2.3	94.6
			1-LR	48	31.3	44.4	25.4	39.6	17.9	38.2	15.7	**94.9**
			2-LR	**54.7**	**37.6**	**52.4**	**33.1**	**48.7**	**25.7**	**48.0**	**23.6**	94.5
	C100	R50	N-LR	37.2	29.9	34.1	24.6	29.8	15.9	34.1	13.3	**85.8**
			1-LR	**45.3**	**38.7**	**43.7**	**35.8**	**40.9**	**31.5**	**40.0**	**29.8**	**85.8**
		R18	N-LR	30.6	23.2	26.4	16.9	20.5	7.42	18.4	5.1	84.1
			1-LR	**34.5**	**25.4**	**31.3**	**20.2**	**27.3**	**13.1**	**25.7**	**10.8**	**84.2**
			2-LR	33.82	24.37	30.9	19.1	26.8	11.83	25.41	9.9	84
Black Box	C10	R50	1-LR	64.7	56.8	59.0	47.5	51.2	28.0	48.3	20.6	95.4
		R18	1-LR	66.6	60.8	61.1	51.0	52.2	31.52	49.8	23.6	94.9
			2-LR	68.0	62.5	62.3	53.4	53.8	33.5	50.8	25.8	94.5
	C100	R50	1-LR	52.4	46.2	48.1	38.8	42.0	25.4	48.1	20.9	85.8
		R18	1-LR	53.0	48.6	47.9	41.0	41.1	26.3	38.7	20.4	84.2
			2-LR	51.3	47.2	46.9	39.9	39.5	24.3	37.2	19.2	84

Table 6.1: Adversarial Test Accuracy against an ℓ_∞ constrained PGD adversary with the ℓ_∞ radius bounded by ϵ and the number of attack steps bounded by τ. R50 and R18 denotes ResNet50 and ResNet18 respectively. C10 and C100 refer to CIFAR10 and CIFAR100 (Coarse labels) respectively. Black box attacks are generated using the N-LR model.

on the negative loss function [172].

We also consider the **black-box** version of each of the aforementioned adversarial attacks where the noise is constructed using N-LR. This is to avoid situations where LR might be at an advantage due to the low-rank structure that might enforce a form of gradient masking [256].

Robustness to Adversarial Attacks In Table 6.1, we measure the adversarial test accuracy of ResNet18 and ResNet50 models trained on CIFAR10 and CIFAR100 respectively. The adversary used here is an L_∞ PGD adversary (or IFGSM) that has two main constraints — the ℓ_∞ radius and the number of attack-steps the PGD algorithm can take. The ℓ_∞ radius is chosen from $\{8/255, 10/255, 16/255, 20/255\}$ with either 7 or 20 attack steps of PGD. This represents a wide variety of severity in the attack model and our LR model performs much better than the N-LR model in all the settings including the black-box settings. For example, in the case of a

			Adversarial Test Accuracy(%)								Test Accuracy (%)
L_∞ radius			8/255		10/255		16/255		20/255		
Attack iterations			7	20	7	20	7	20	7	20	
White Box	R50	N-LR	27.8	21.8	25.2	17.7	21.1	17.7	19.4	7.6	77.2
		1-LR	**28.8**	**23.6**	**26.8**	**21.4**	**24.4**	**17.8**	**23.5**	**16.5**	**77.7**
Black Box	R50	1-LR	38.3	32.8	34.3	26.4	28.9	15.0	27.0	11.7	-

Table 6.2: Adversarial Test Accuracy against a ℓ_∞ constrained PGD adversary with the ℓ_∞ radius bounded by ϵ and the number of attack steps bounded by τ on CIFAR100 fine labels for ResNet50.

	Adversarial Test Accuracy(%)					
L_∞ radius	$^8/_{255}$		$^{16}/_{255}$		$^{20}/_{255}$	
Att. iter	7	20	7	20	7	20
N-LR	43.1	31.0	31.2	7.8	28.9	4.5
SNIP	29.4	14.5	18.5	1.3	16.2	0.4
SRN	47.8	37.6	39.8	21.3	37.5	18.4
LR (Ours)	**79.1**	**78.5**	**77.9**	**77.0**	**77.1**	**76.6**

Table 6.3: Robustness of other regularisation/compression approaches to Adversarial attacks.

white-box attack with $\ell_\infty = \frac{16}{255}$ and 20 attack steps on a ResNet-50 model trained on CIFAR10, the LR ResNet model is nearly **10 times more accurate** than the N-LR model.

In Table 6.2, we show the adversarial test accuracy for varying perturbation budgets for ResNet50 trained on the fine labels of CIFAR100. LR models, not only have the best adversarial test accuracies but also the best natural test accuracies.

The above results indicate that LR provides low-rank representations that are robust to adversarial perturbations and also provide modest improvements on the test accuracy.

In Table 6.3, we also compare our LR with other methods that reduce some form of intrinsic dimensions of the parameter space. SNIP [158], a pruning technique, increases the sparsity of the parameter and SRN [232] reduces the (stable) rank and spectral norm of the parameters. We use the best hyper-parameter settings suggested for these approaches in their manuscripts. For a description of these methods and other related approaches please refer to Appendix 6.5. Our method performs much better than all these methods indicating that reducing the dimensionality of the

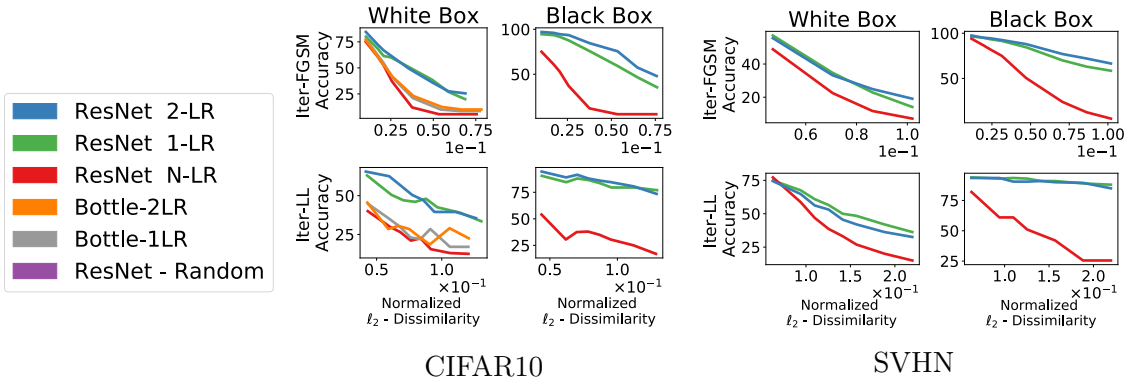


Figure 6.5: Adversarial accuracy of ResNet18 against the magnitude of perturbation (measured by ρ) on CIFAR10 and SVHN.

representation space is much more effective than doing so for the parameter space when it comes to the robustness of the network.

Accuracy vs the amount of adversarial noise Next, we compare the change in accuracy of adversarial classification with respect to the actual amount of noise added (as opposed to the perturbation budget as in Table 6.1). The amount of noise added can be measured using the normalized L_2 dissimilarity score (ρ), defined as:

$$\rho = \mathbb{E}\left[\|\mathbf{x}_{\mathrm{a}} - \mathbf{x}_{\mathrm{d}}\|_2 / \|\mathbf{x}_{\mathrm{d}}\|_2\right],$$

where $\mathbf{x}_d$ and $\mathbf{x}_a$ are the clean and adversarially perturbed samples, respectively. The normalized L_2 dissimilarity score ρ measures the magnitude of the noise [186] in the input corresponding to a certain adversarial misclassification rate[5].

Here we also consider another model, for comparison, we call the *Bottle-LR* model. It contains an *explicit bottleneck* low-rank layer, rather than LR-layer, which is essentially a fully connected layer without any non-linear activation where the weight matrix $\bar{\mathbf{W}} \in \mathbb{R}^{q\times q}$ is parameterised by $\bar{\mathbf{W}}_l \in \mathbb{R}^{q\times r}$, where $r \leq q$, so that $\bar{\mathbf{W}} = \bar{\mathbf{W}}_l \bar{\mathbf{W}}_l^\top$. Note, by design, it can not have a rank greater than r.

Figure 6.5 shows that as the noise increases, the accuracy of N-LR models decreases much faster than the LR models. Specifically, to reach an adversarial mis-

[5]Similar to the setting in Kurakin et al. [153], the noise is added for a pre-determined number of steps.

	Model	ρ [DeepFool]	ρ [Iter-LL-FGSM]	ρ [Iter-FGSM]
White Box	2-LR	$\mathbf{1.8 \times 10^{-1}}$	9.8×10^{-2}	$\mathbf{7.6 \times 10^{-2}}$
	1-LR	1.7×10^{-1}	$\mathbf{1.1 \times 10^{-1}}$	6.0×10^{-2}
	N-LR	1.6×10^{-2}	2.4×10^{-2}	2.1×10^{-2}
Black Box	2-LR	$\mathbf{5.5 \times 10^{-2}}$	$\mathbf{2.0 \times 10^{-1}}$	$\mathbf{7.5 \times 10^{-2}}$
	1-LR	4.7×10^{-2}	1.8×10^{-1}	5.6×10^{-2}

Table 6.4: Minimum perturbation required for 99% Adversarial Misclassification by ResNet18 models on CIFAR10. Table 6.5 shows the ℓ_∞ perturbations used. The values of the perturbation budget also show that the minimum perturbation required for 99% misclassification is much higher for LR models than N-LR models.

classification rate of 50%, our models require about twice the noise as the N-LR or Bottle-LR. *For* all *kinds of attacks we considered, LR models consistently outperform N-LR and Bottle-LR.*

Even though the rank constraint of Bottle-LR is the same as LR models, and they are placed at the same position as the LR layers, the inferior performance of Bottle-RL (sometimes even worst than N-LR) can be explained by the fact that the explicit bottleneck has a *multiplicative* effect on the back-propagated gradients, whereas, LR's impact on gradients is *additive*. With a bottleneck layer, the gradient of any $\mathbf{W}_j$ before the bottleneck layer is $\partial\mathcal{L}/\partial\mathbf{W}_j = (\partial\mathcal{L}/\partial\mathbf{z}_l)\bar{W}(\partial\mathbf{a}_{l-1}/\partial\mathbf{W}_j)$ where $\mathbf{a}_l$ and $\mathbf{z}_l$ are the activations and the pre-activations of the l^{th} layer respectively. Thus, especially during the early stages of training when $\bar{\mathbf{W}}$ is not yet learned, important directions in $(\partial\mathbf{a}_{l-1}/\partial\mathbf{W}_j)$ can be cancelled out due to the low-rank nature of $\bar{\mathbf{W}}$, thus making $\partial\mathcal{L}/\partial\mathbf{W}_j$ uninformative. In the case of LR, the gradients from $\mathcal{L}_c$ (depending on W) and $\mathcal{L}$ are additive and thus the low-rank $\mathbf{W}$ only affects the gradients from $\mathcal{L}$ additively. The auxiliary loss $\mathcal{L}_c$ has a less direct impact on early training (in terms of classification error). We believe this relative "smoothness" of our approach is the reason why it has a better performance on these other tasks.

Minimum adversarial perturbation for 99% **misclassification** Our next experiment is along the lines of that reported in Moosavi-Dezfooli et al. [186]. Table 6.4 shows the average minimum perturbation (measured by ρ) required to make the classifier misclassify more than 99% of the adversarial examples, constructed from a uniformly sampled subset of the test set. We use the deepfool algorithm described in

	Model	ϵ [Iter-LL-FGSM]	ϵ [Iter-FGSM]
White Box	2-LR	4×10^{-2}	2×10^{-2}
	1-LR	6×10^{-2}	1×10^{-2}
	N-LR	1×10^{-2}	1×10^{-2}
Black Box	1-LR	8×10^{-2}	1×10^{-2}
	2-LR	1×10^{-1}	1×10^{-2}

Table 6.5: Value for ϵ required for Adversarial Misclassification corresponding to Table 6.4.

Algorithm 2 in Moosavi-Dezfooli et al. [186] (also discussed in Section 3.2.2). The algorithm returns the minimum perturbation $r(\mathbf{x})$ required to make the classifier misclassify the point $\mathbf{x}$. The L_2 dissimilarity is obtained by calculating $\rho = \frac{r(\mathbf{x})}{\|\mathbf{x}\|_2}$. For our experiments, we used the publicly available code of DeepFool. [6].

Even under this scheme of attacks, our models perform better than N-LR as LR models require 4 to 11 times the amount of noise required by N-LR models to be fooled by adversarial attacks (see Table 6.4). This can also be visualised in Figure 6.6. The adversarial images for 2-LR and 1-LR are noticeably much more perturbed than N-LR. An interesting observation is that the values of L_2 dissimilarity in Table 6.4 are lower than those in Figure 6.5 though the attacks have a higher rate of success. The essential difference between the attacks in Figure 6.5 and Table 6.4 is in the number of iterations for which the updates are executed while creating the attack. In Figure 6.5, the step is executed for a fixed number of steps whereas, in Table 6.4, the updates are executed until the classifier makes a mistake. This can be explained by our discussion around Figure 3.1 where we show that attacks that stop adding adversarial perturbations upon successful misclassification are more powerful than those that add the perturbation for a fixed number of steps.

6.4.3 Noise stability

To gain some understanding of this visibly better adversarial robustness of LR models, in this section, we study the noise stability behaviour of LR models in detail. Specifically, we show that LR models (and their representations) are significantly more stable to input perturbations at test time even when training is performed using clean data.

[6] `https://github.com/LTS4/DeepFool/blob/master/Python/deepfool.py`

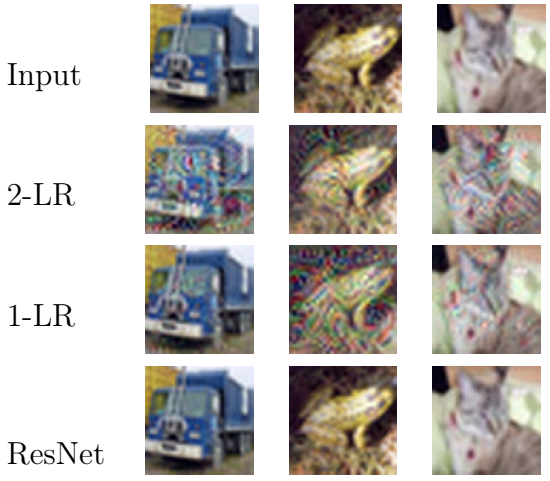


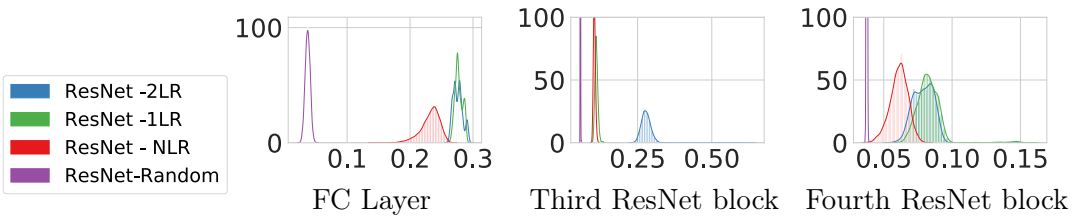
Figure 6.6: Adversarial images using DeepFool against different classifiers. The images against low-rank (LR) classifiers require more perturbations and are noticeably different.

Pert. Prob. (p)		0.4	0.6	0.8	1.0
R50	N-LR	69.7	26.1	12.6	11.3
	1-LR	**75.1**	**34.2**	**15.8**	**13.0**
R18	N-LR	57.7	27.3	13.0	7.2
	1-LR	**75.1**	33.0	15.2	11.0
	2-LR	74.1	**35.5**	**16.4**	**11.5**

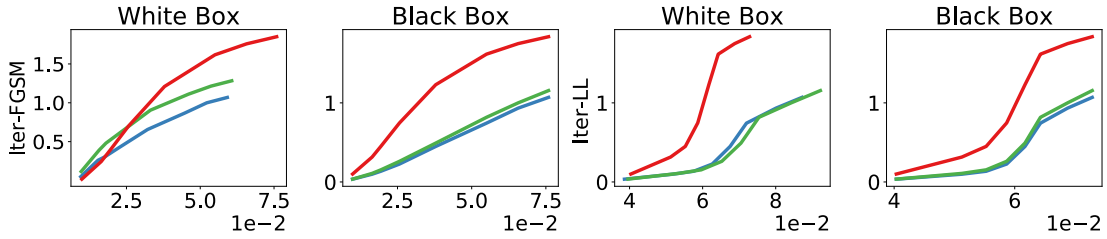
Table 6.6: Test accuracy of ResNet50 (R50) and ResNet18(R18) to Gaussian noise $\mathcal{N}(0, {}^{128}/_{255})$ introduced at each pixel with probability p. Evaluated on CIFAR10.

Random Pixel Perturbations In Table 6.6, we measure the test accuracy when the input is perturbed with random additive noise. Specifically, for a given pixel and a given *pixel perturbation probability* $p \in \{0.4, 0.6, 0.8, 1.0\}$, we toss a biased coin (with bias p) and if heads, add to the pixel a Gaussian noise drawn from $\mathcal{N}(0, {}^{128}/_{255})$. This is done for all the pixels in the test-set and the test accuracy is measured over this perturbed dataset. For varying levels of perturbation, Table 6.6 shows that LR models are significantly more stable to Gaussian noise than N-LR. Our experiments indicate that learning a model that cancels out irrelevant directions in the representations suppresses the propagation of the input noise in a way to reduce its effect on the output of the model. Interestingly, the level of Gaussian noise seems to not vary the test accuracy as much as the value of p does.

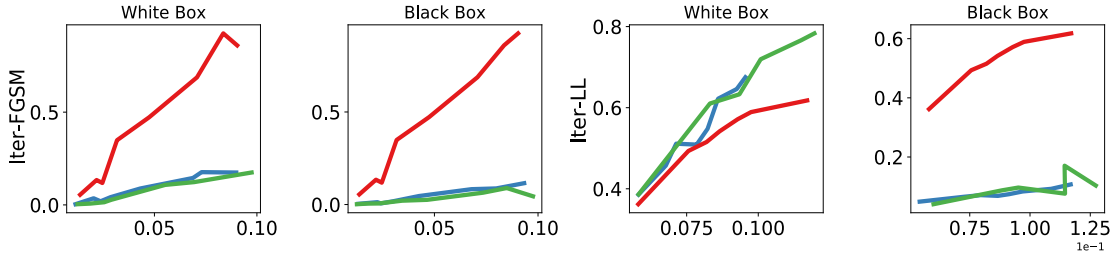
Stability of representations In Figures 6.7(b) and 6.7(c), we show how the input adversarial perturbations propagate and impact the feature space representations. Specifically, for a given adversarial perturbation δ to an input $\mathbf{x}$, the x-



(a) Layer cushion of ResNet layers. (Right is better)



(b) Adversarial Perturbation in Input (x-axis) and Representation (y-axis) Space for CIFAR10. (Lower is better)



(c) Adversarial Perturbation in Input (x-axis) and Representation (y-axis) Space for SVHN. (Lower is better)

Figure 6.7: Layer cushion of the last fully connected layer, last convolution layer of the third residual block, and last convolution layer of the fourth residual blocks are plotted, respectively, in Figure 6.7(a). Noise-sensitivity of ResNet-18 trained on is showed via plotting the propagation of perturbation from the input to the representation space for CIFAR10 in (Figure 6.7(b)) and SVHN in Figure 6.7(c).

axis is the normalized L_2 dissimilarity score in the input space i.e. $\|\delta\|^2/\|\mathbf{x}\|^2$ and the y-axis represents the corresponding quantity in the representation space i.e. $\left\|\mathrm{f}_\ell^-(\mathbf{x}+\delta)-\mathrm{f}_\ell^-(\mathbf{x})\right\|^2/\left\|\mathrm{f}_\ell^-(\mathbf{x})\right\|^2$. The representations $f_\ell^-(.)$ here are taken from before the last fully connected layer. As our experiments suggest, the LR model significantly attenuates the adversarial perturbations thus making it harder to fool the softmax classifier. This observation further supports the increased robustness of LR.

Layer cushion Arora et al. [16] gives empirical evidence that deep networks are stable towards injected Gaussian noise and use a variation of this noise stability property to derive more realistic generalisation bounds. They capture the noise-sensitivity through a term called layer cushion, which we define in Definition 14.

It is measured for each example in the training dataset. The higher the value of the layer cushion, the better is the generalisation ability of the model. Figure 6.7(a) shows histogram plots of the distribution of this term for the examples in CIFAR10 for four models—2-LR, 1-LR, N-LR and a *random network* (randomly initialised, no training done) with the same architecture. As Figure 6.7(a) shows, the histograms of the LR models are to the right of the N-LR model, which is further to the right of the randomly initialised network thus indicating that LR models have the highest layer cushion.

6.4.4 Discriminative properties of embeddings

Experiments in the previous section showed that our algorithm induces a low-rank structure in the activation space and that these low-rank activations are significantly more stable to input perturbations. Here we inspect the discriminative power of these embeddings.

Hybrid Max-Margin models: For the experiments in this section, we will use modified versions of the original models which we will refer to as *hybrid max-margin models*. To convert a base model to a hybrid model, learned representations are first generated using the original trained base model on the training dataset. Then a max-margin classifier (such as SVM) is trained on these learned representations to classify the original label. In some of the experiments, before training the max-margin classifier, the learned representations are projected onto a low dimensional space by performing PCA on this set of learned representations to classify the original label. This dimension of projection will be referred to as *dim* when the results are presented. At test time, the original trained model is used to first obtain a representation, if necessary, and then projected using the learnt PCA projection matrix, and is then classified using the learnt max-margin linear classifier. The linear max-margin model is trained using SGD with hinge loss and 0.01 L_2 regularisation coefficient. The learning rate is decreased per iteration as $\eta_t = \frac{\eta_0}{(1+\alpha t)}$ where η_0 and α are set by certain heuristics [7].

[7] `https://cilvr.cs.nyu.edu/diglib/lsml/bottou-sgd-tricks-2012.pdf`

	Accuracy(%)	
Models	Coarse	Fine
R50 1-LR	**78.1**	48
R50 N-LR	75.6	**52**
R50 Bottle-1LR	76	38

Table 6.7: Transfer learning on CIFAR-100 using LR and NLR models

Transfer learning As the intrinsic dimensionality of the representations are decreased, it is plausible that the representations only store the most necessary information to solve the task at hand and ignore all other information from the input. If that were true, then representations learned for one task cannot be successfully used for another related but slightly different task. To test whether the learned representations can be used in a different task, we conduct a transfer learning exercise where embeddings generated from a ResNet-50 model (after the fourth resnet block), trained on the coarse labels of CIFAR-100, are used to predict the fine labels of CIFAR-100. A set of ResNet-50 hybrid max-margin classifiers is trained for this purpose using these 2048 dimensional embeddings.

The accuracy of the hybrid classifiers is reported in Table 6.7. The results of the same experiment, when the hybrid model is trained on the coarse labels are reported in the second column of Table 6.7. Surprisingly, the low-rank model performs well at all in this experiment as one would expect that all information in the representations that are not strictly required in the classification of the original task are discarded from the model.

Table 6.7 shows that the LR model suffers a small loss of 4% in accuracy as compared to the N-LR model when its embeddings are used to train a max-margin classifier for predicting fine labels. It should be noted that the accuracy of the LR model actually increases when the max-margin classifier is trained to perform the original task, i.e. classifying the coarse labels. On the other hand, the Bottle-LR model suffers a loss of 14% in accuracy compared to the N-LR model and shows no significant advantage in the original task either.

Clusters of low dimensional embeddings Figure 6.8 shows the two-dimensional projections of the 2048 dimensional embeddings obtained from ResNet-50-LR and

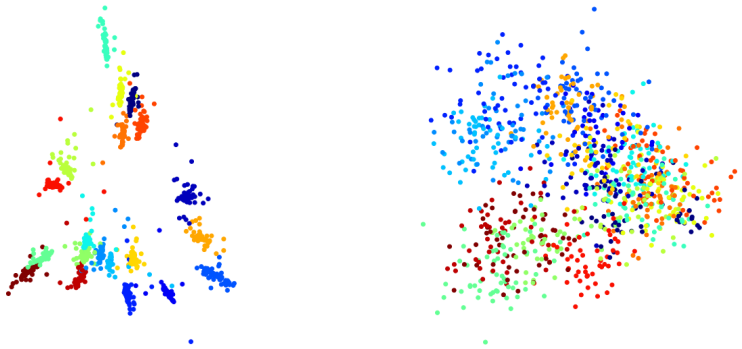


Figure 6.8: 2-D PCA projection of representations from ResNet50 trained on coarse labels of CIFAR 100 with (left) and without (right) low-rank constraints, coloured according to the original 20 coarse labels.

ResNet-50-N-LR. The colouring is done according to the coarse labels of the input. We can see that the clusters are more separable in the case of the model with LR-layer than the model without, which gives some insight into why a hybrid max-margin classifier might perform better for the LR model than the N-LR model. Thus, the representations of the low-rank model are more discriminative in the sense that for the low-rank representations there are low dimensional linear classifiers that can classify the dataset with a higher margin than the vanilla models.

6.4.5 Compression of model and embeddings

Further, due to the low dimensionality of the representation space, these learned representations can be compressed without losing their discriminative power. Among other things, we show that low dimensional projections of our embeddings, *with a size of less than* 2% *of the original embeddings*, can be used for classification with significantly higher accuracy than similar-sized projections of embeddings from a model trained without our training modification (N-LR) on the CIFAR100 dataset.

Representation compression: In the first experiment, reported in Table 6.8(a), we trained two ResNet-50 hybrid max-margin models -with and without the LR-layer respectively- on the 20 super-classes of CIFAR-100. As our objective here is to see if the embeddings and their low dimensional projections could be effectively used for discriminative tasks, we used PCA, with standard pre-processing of scaling the input, to project the embeddings onto a low dimensional space and then trained

a linear maximum margin classifier on it.

Table 6.8(a) shows that even with sharply decreasing embedding dimension, the hybrid model trained using the LR preserves the accuracy significantly more so than N-LR. *Even with a* 5*-dimensional embedding (400x compression), the LR model loses only* 6% *in accuracy, but the N-LR model loses* 27%. In Table 6.9, we show that compressed representations are also more robust to adversarial attacks.

Model	Dim	Accuracy(%)
R50 1-LR	$2k$	**78.1**
R50 N-LR	$2k$	75.6
R50 1-LR	10	**76.5**
R50 N-LR	10	68.4
R50 1-LR	5	**72**
R50 N-LR	5	48

(a) Representation before FC layer, trained on CIFAR-100. Original dimension is $2k$.

Model	Dim	Accuracy(%)
R18 2-LR	$16k$	**91.14**
R18 N-LR	$16k$	90.7
R18 2-LR	20	**88.5**
R18 N-LR	20	76.9
R18 2-LR	10	**75**
R18 N-LR	10	61.7

(b) Representation before last ResNet block, trained on CIFAR10. Original dimension is $16k$.

Table 6.8: *Dim* is the size of the *compressed embedding* on which linear classifier is trained.

Model compression: A consequence of forcing the activations of the ℓ^{th} layer of the model to lie in a low dimensional subspace with minimal reconstruction error, is that a simpler model can replace the latter parts of the original model without significant reduction in accuracy. Essentially, if we train the hybrid max-margin classifier on the representations obtained from after the third ResNet block, we can replace the entire fourth ResNet block and the last FC layer (with 8.4M parameters) with a small linear classifier with only 0.02 times the number of parameters.

In this experiment, we trained two ResNet-18 hybrid max-margin classifiers- with and without the LR-layer respectively- on CIFAR-10. The representations were obtained from before the fourth ResNet block and had a dimension of 16,384. This yields a significant reduction in model size at the cost of a very slight drop in accuracy ($< 1\%$). The second benefit is that as the low dimensional embeddings still retain most of the *discriminative* information, the inputs fed to the linear model also have a small number of features.

6.4.6 Adversarial robustness of maximum margin model

	R18	DFL	ILL	IFGSM
White Box	2-LR	**0.43**	**0.55**	**0.55**
	1-LR	0.38	0.35	0.48
	N-LR	0.01	0.04	0.02
Black Box	2-LR	**0.44**	**0.50**	**0.48**
	1-LR	0.29	0.31	0.33

Table 6.9: Accuracy of classification of adversarial examples, constructed by attacking ResNet18, by ResNet18 Max-Margin Classifiers.

Robustness of Max-Margin Classifiers Finally, we show that the features learned by our models are inherently more linearly discriminative i.e. there exists a linear classifier that can be used to classify these features with a wide margin. To this end, in Table 6.9, we show that for LR models, the max-margin hybrid models are significantly more robust to adversarial attacks than the corresponding original models. Hybrid max-margin models with LR-layers are particularly more robust than hybrid max-margin models without LR-layers against adversarial attacks. Specifically, as seen in Table 6.9, *a hybrid model with an* LR-layer *correctly classifies* 50% *of the examples that had fooled the original classifier while for a similar amount of noise, an N-LR hybrid model has negligible accuracy.*

We train a variety of hybrid max-margin models with ResNet18-1-LR, ResNet18-2-LR, and ResNet18-N-LR along with black box versions of the same. Then we generate adversarial examples for all three attacks (both black box and white box) on the trained ResNet models (not the hybrid models). Then we use these adversarial examples to attack the corresponding max-margin models

To perform a fair comparison with the hybrid ResNet18-N-LR, it is essential to add a similar amount of noise to generate the examples for the hybrid ResNet18-N-LR as is added to hybrid ResNet18-1-LR. The adversarial examples are hence generated by obtaining the gradient using ResNet18-N-LR but stopping the iteration only when the adversarial example could fool ResNet18-1-LR.

6.5 Alternative algorithms for low-rank representations

In this section, we discuss a few different alternative algorithms that could be potential alternatives for obtaining low-rank representations.

Low-rank weights

With respect to compression, it is natural to look at low-rank approximations of network parameters [67, 128]. By factorizing the weight matrix $\mathbf{W}$ as the product of a wide and a tall matrix, we can get low-rank *pre-activations*. This however does not lead to low-rank *activations* as demonstrated both mathematically (by the counter-example below) and empirically.

Mathematical counter-example : Consider a rank 1 *pre-activation* matrix $\mathbf{A}$ and its corresponding *post-activation*(ReLU) matrix as below. It is easy to see that the rank of *post-activation* has increased to 2.

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix} \qquad \text{Relu}(\mathbf{A}) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Empirical Result: To see if techniques for low-rank approximation of network parameters like Denton et al. [67] would have produced low-rank activations, we experimented by explicitly making the *pre-activations* low-rank using SVD. Our experiments showed that despite setting a rank of 100 to the *pre-activation* matrix, the *post-activation* matrix had full rank. Though all but the first hundred singular values of the *pre-activation* matrix were set to zero, the *post-activation* matrix's 101^{st} and 1000^{th} singular values were 49 and 7.9 respectively, and its first 100 singular values explained only 94% of the variance.

Theoretically, a bounded activation function lowers the Frobenius norm of the *pre-activation* matrix i.e. the sum of the squared singular values. However, it also causes a smoothening of the singular values by making certain zero singular values non-zero to compensate for the significant decrease in the larger singular values.

This leads to an increase in rank of the *post-activation* matrix.

In Table 6.3, we compare against SRN [232] a simple algorithm for reducing the stable rank (a softer version of rank) of linear layers in neural networks and observe that the increase in adversarial robustness is not as high as provided by LR models.

Nuclear norm

Nuclear norm regularisation is a convex relaxation to the hard rank regularisation approach that we have adopted. Nuclear norm can be regularised by adding the nuclear norm of $\mathbf{A}_\ell$ to the loss function, i.e. minimizing $\mathcal{L}(\mathbf{X}, \mathbf{Y}; \theta, \phi) + \lambda \|\mathbf{A}_\ell\|_*$.

However, there are a few problems with it. First, it runs into the same problem as the hard rank minimisation - a) unfeasible to optimize for the large whole activation matrix, b) unclear why batch-wise optimisation should ensure low-rank for the entire dataset, and c) sensitivity towards the hyper-parameter. Its sensitivity to hyper-parameter arises for reasons similar to the hyper-parameter sensitivity of ridge-regression as discussed at the end of Section 6.3.

We performed batch-wise nuclear norm minimisation for all $\lambda \in \{1, 0.1, 0.01, 0.001, 0.0001\}$ and observed that only 0.001 gave comparable performance to LR. All other settings gave either a trivial test accuracy or much worse adversarial robustness thus showing its extreme sensitivity to hyper-parameters, unlike our method.

6.6 Additional figures and tables in appendix

Figure E.1 compares the layer cushion of the four different residual blocks of a ResNet18 on CIFAR10. Note that only 2-LR shows an increased cushion in Layer 3 whereas both 1-LR and 2-LR have higher cushions in all other layers. Similarly, Figure E.2 plots the layer cushion of a ResNet-18 trained on SVHN.

7. Improving Calibration via Focal Loss

In the previous chapters, we looked at vulnerabilities of deep neural networks arising from poor generalisability (see Chapter 4) and adversarial robustness (see Chapters 5 and 6) of neural networks. Another common issue faced by many state-of-the-art neural networks is that they are poorly calibrated. Neural networks, used for multi-class classification, output a distribution over the label classes where each probability value is supposed to indicate the likelihood of that class label being the correct one. In many state-of-the-art neural networks, the probability values they associate with the predicted class labels overestimate the likelihoods of those class labels being correct in the real world. The underlying cause is hypothesised to be that the high complexity of these networks leaves them vulnerable to overfitting on the negative log-likelihood (NLL) loss they conventionally use during training [99]. As discussed in Section 3.3, multiple solutions have been proposed to tackle this problem including the popularly used *Temperature Scaling*.

In this chapter, we propose a technique for improving network calibration that works by replacing the cross-entropy loss conventionally used when training classification networks with the focal loss proposed by [165]. We observe that unlike cross-entropy, which minimises the KL divergence between the predicted (softmax) distribution and the target distribution (one-hot encoding in classification tasks) over classes, focal loss minimises a regularised KL divergence between these two distributions i.e. focal loss minimises the KL divergence whilst *increasing the entropy* of the predicted distribution over the class labels, thereby preventing the model from becoming overconfident.

The intuition behind using focal loss is to weight the gradient updates during training more towards samples for which it is currently predicting a low probability for the correct class. This helps to avoid reducing the NLL on samples where it is already predicting a high probability for the correct class as that is liable to lead to NLL overfitting and thereby miscalibration [99]. We show in Section 7.3 that focal

loss can be seen as *implicitly* regularising the weights of the network during training by causing the gradient norms for confident samples to be lower than they would have been with the cross-entropy loss, which we would expect to reduce overfitting and improve the network's calibration.

7.1 Main contributions

Overall, we make the following contributions:

1. We study the link that Guo et al. [99] observed between miscalibration and NLL overfitting; show that NLL overfitting is associated with the predicted distributions for misclassified test samples becoming peakier as the learning algorithm increases the magnitude of the network's weights to reduce the training NLL.

2. We propose the use of focal loss for training better-calibrated networks and provide both theoretical and empirical justifications for our approach. In addition, we provide a principled method for automatically choosing the hyperparameter γ for each data point during training.

3. We show, via experiments on a variety of classification datasets and network architectures, that DNNs trained with focal loss are more calibrated than those trained with cross-entropy loss (both with and without label smoothing), MMCE, and Brier loss.

7.2 Understanding the cause of miscalibration

Problem Formulation Let $\mathcal{S}_N = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$ denote a dataset consisting of N samples from a data distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, where for each sample i, $\mathbf{x}_i \in \mathcal{X}$ is the input and $y_i \in \mathcal{Y} = \{1, 2, ..., K\}$ is the target class label. Let $\hat{p}_{i,y} = f_\theta(\mathbf{x}_i)[y]$ be the probability that a neural network f with model parameters θ assigns to a class y on a given input $\mathbf{x}_i$. The class that f predicts for $\mathbf{x}_i$ is computed as $\hat{y}_i = \text{argmax}_{y \in \mathcal{Y}}\ \hat{p}_{i,y}$, and the predicted confidence is computed as $\hat{p}_i = \max_{y \in \mathcal{Y}}\ \hat{p}_{i,y}$. The network is said to be *perfectly (weakly) calibrated* when, for each sample $(\mathbf{x}_j, y_j) \in \mathcal{S}_N$, the confidence $\hat{p}_j$ is equal to the model accuracy

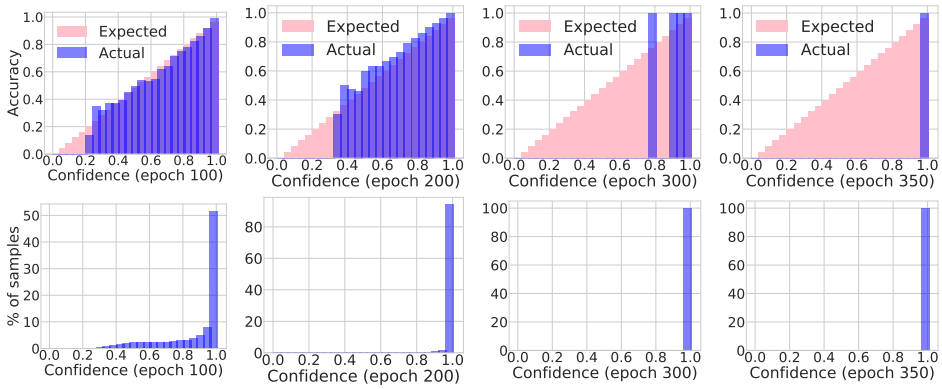


Figure 7.1: The confidence values for training samples at different epochs during the NLL training of a ResNet-50 on CIFAR-10 (see Section 7.2). Top row: reliability plots using 25 confidence bins; bottom row: % of samples in each bin. As training progresses, the model gradually shifts all training samples to the highest confidence bin. Notably, it continues to do so even after achieving 100% training accuracy by the 300 epoch point.

$\mathbb{P}\left[\mathcal{Y} = \hat{y}_j \mid f_\theta\left(\mathbf{x}_j\right)\left[\hat{y}_j\right] = \hat{p}_j\right]$, i.e. the probability that the predicted class is correct. For instance, of all the samples to which a perfectly calibrated neural network assigns a confidence of 0.8, 80% should be correctly predicted.

We will use the various metrics described in Section 3.3.2 to measure calibration in our experiments in this chapter. This includes Expected Calibration Error (ECE), Classwise Expected Calibration Error, Adaptive ECE (AdaECE), and reliability plots.

We now discuss why large networks, despite achieving low classification errors on well-known datasets, tend to be miscalibrated. A key empirical observation made by [99] was that poor calibration of such networks appears to be linked to overfitting on the negative log-likelihood (NLL) during training. In this section, we discuss what we mean by NLL overfitting and further inspect how NLL overfitting impacts calibration.

For the analysis, we train a ResNet-50 network on CIFAR-10 with state-of-the-art performance settings as discussed in Appendix B. We minimise cross-entropy loss (a.k.a. NLL) $\mathcal{L}_c$, which, in a standard classification context, is $-\log \hat{p}_{i,y_i}$, where $\hat{p}_{i,y_i}$ is the probability assigned by the network to the correct class y_i for the ith sample. Note that the NLL is minimised when for each training sample i, $\hat{p}_{i,y_i} = 1$, whereas the classification error is minimised when $\hat{p}_{i,y_i} > \hat{p}_{i,y}$ for all $y \neq y_i$. This indicates that even when the classification error is 0, the NLL can be positive, and

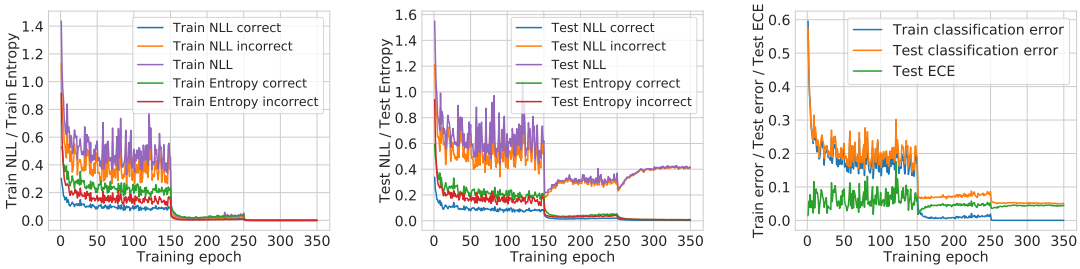


Figure 7.2: Metrics related to calibration plotted whilst training a ResNet-50 network on CIFAR-10.

the optimisation algorithm can still try to reduce it to 0 by further increasing the value of $\hat{p}_{i,y_i}$ for each sample.

To empirically observe this, we divide the confidence range $[0, 1]$ into 25 bins, and present reliability plots computed on the training set at training epochs 100, 200, 300 and 350 (see the top row of Figure 7.1). In Figure 7.1, we also show the percentage of samples in each confidence bin. It is quite clear from these plots that over time, the network gradually pushes all of the training samples towards the highest confidence bin. Furthermore, even though the network has achieved 100% accuracy on the training set by epoch 300, it still pushes some of the samples lying in lower confidence bins to the highest confidence bin by epoch 350.

To study how miscalibration occurs during training, we plot the average NLL for the train and test-sets at each training epoch in Figures 7.2(a) and 7.2(b). We also plot the average NLL and the entropy of the softmax distribution produced by the network for the correctly and incorrectly classified samples. In Figure 7.2(c), we plot the classification errors on the train and test datasets, along with the test-set ECE.

Curse of misclassified samples: Figures 7.2(a) and 7.2(b) show that although the average train NLL (for both correctly and incorrectly classified training samples) broadly decreases throughout training, after the 150^{th} epoch (where the learning rate drops by a factor of 10), there is a marked rise in the average test NLL, indicating that the network starts to overfit on average NLL. This increase in average test NLL is caused only by the incorrectly classified samples, as the average NLL for the correctly classified samples continues to decrease even after the 150^{th} epoch. This phenomenon is referred to as NLL overfitting. We also observe that after epoch

150, the test-set ECE rises, indicating that the network is becoming miscalibrated. This corroborates the link between NLL overfitting and miscalibration observation in [99].

Peak at the wrong place: We further observe that the entropies of the softmax distributions for both the correctly and incorrectly classified *test* samples decrease throughout training (in other words, the distributions get peakier). This observation, coupled with the one we made above, indicates that *for the wrongly classified test samples, the network gradually becomes more and more confident about its incorrect predictions.*

Weight magnification: The increase in confidence of the network's predictions can happen if the network increases the norm of its weights W to increase the magnitudes of the logits. In fact, the cross-entropy loss is minimised when for each training sample i, $\hat{p}_{i,y_i} = 1$, which is possible only when $||W|| \to \infty$. Cross-entropy loss thus inherently induces this tendency of weight magnification in neural network optimisation. The promising performance of weight decay [99] (regularising the norm of weights) on the calibration of neural networks can perhaps be explained using this. This increase in the network's confidence during training is one of the key causes of miscalibration.

7.3 Improving calibration using focal loss

As discussed in the previous section, overfitting on NLL, which is observed as the network grows more confident on all of its predictions irrespective of their correctness, is strongly related to poor calibration. One cause of this is that minimising the cross-entropy loss function minimises the difference between the softmax distribution and the ground-truth one-hot encoding for all samples, irrespective of how well a network classifies individual samples. In this chapter, we study an alternative loss function, popularly known as *focal loss* [165], that tackles this by weighting loss components generated from individual samples by how well the model classifies each of them. For classification tasks where the target distribution is one-hot encoding, it is defined as $\mathcal{L}_f = -(1 - \hat{p}_{i,y_i})^\gamma \log \hat{p}_{i,y_i}$, where γ is a user-defined hyperparameter.

Remark 8. *We note in passing that, unlike cross-entropy loss, focal loss in its general form is not a proper loss function, as minimising it does not always lead to the predicted distribution $\hat{p}$ being equal to the target distribution q. However, when q is a one-hot encoding (as in our case, and for most classification tasks), minimising focal loss does lead to $\hat{p}$ being equal to q.*

7.3.1 Focal loss as a regularised Bregman divergence

We know that the cross-entropy loss forms an upper bound on the KL-divergence between the target distribution q and the predicted distribution $\hat{p}$, i.e. $\mathcal{L}_c \geq \mathrm{KL}(q||\hat{p})$, so minimising cross-entropy minimises $\mathrm{KL}(q||\hat{p})$. Interestingly, a general form of focal loss can be shown to be an upper bound on the regularised KL-divergence, where the regulariser is the negative entropy of the predicted distribution $\hat{p}$, and the regularisation parameter is γ, the hyperparameter of focal loss.

Theorem 5 (Focal Loss minimises a regularised Bregman divergence)**.** *Let q and $\hat{p}$ denote the target class probabilities and predicted posterior class probabilities respectively and $\mathcal{L}_f$ denote the focal loss with parameter γ. Then,*

$$\mathcal{L}_f \geq \mathrm{KL}(q||\hat{p}) + \underbrace{\mathbb{H}[q]}_{constant} - \gamma \mathbb{H}[\hat{p}]. \tag{7.1}$$

Proof in Appendix F.1

Theorem 5 shows that minimising focal loss minimises the KL divergence between $\hat{p}$ and q, whilst simultaneously increasing the entropy of the predicted distribution $\hat{p}$. Thus replacing cross-entropy with focal loss has the effect of adding a maximum-entropy regulariser [209] to the implicit minimisation of the KL-divergence that was previously being performed by the cross-entropy loss. Encouraging the predicted distribution $\hat{p}$ to have higher entropy can help avoid the overconfident predictions produced by neural networks trained with NLL loss (see the 'Peak at the wrong place' paragraph of Section 7.2), and thereby improve calibration.

In the case of one-hot target labels (i.e. Dirac delta distribution for q), the entropy of the target label probabilities $\mathbb{H}[q]$ is equal to 0 and the KL-divergence

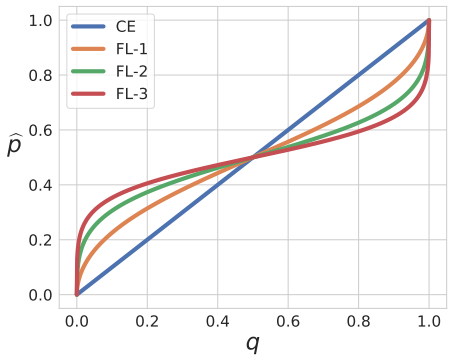


Figure 7.3: Optimal $\hat{p}$ for various values of q. FL-1, FL-2, and FL-3 indicate Focal Loss with $\gamma = 1, 2$, and 3 respectively.

term reduces to $-\log \hat{p}_y$, where y is the ground truth class. So, focal loss maximises $-\log \hat{p}_y - \mathbb{H}[\hat{p}]$ and prefers learning $\hat{p}$ such that $\hat{p}_y$ is assigned a high value (because of the KL term $-\log \hat{p}_y$), but not too high (because of the entropy term), and will ultimately avoid preferring overconfident models (by contrast to cross-entropy loss). We solved the cross-entropy and focal loss equations numerically, i.e. the value of the predicted probability $\hat{p}$ which minimises the loss, for various values of q in a binary classification problem and plotted it in Figure 7.3. As expected, focal loss favours a more entropic solution for $\hat{p}$ that is closer to the uniform distribution. In other words, Figure 7.3 shows that solutions to focal loss (Equation (7.2)) will always have higher entropy than cross-entropy.

$$\text{Solution for Focal Loss: } \hat{p} = \text{argmin}_x \; -(1-x)^\gamma q \log x - x^\gamma (1-q) \log(1-x) \quad 0 \leq x \leq 1 \tag{7.2}$$

7.3.2 Empirical observations on training with focal loss

To analyse the behaviour of neural networks trained on focal loss, we use the same framework as mentioned above and train four ResNet-50 networks on CIFAR-10: one using cross-entropy loss and three using focal loss with $\gamma = 1, 2$, and 3. We plot various training statistics related to these four networks in Figure 7.4. Figure 7.4(a) shows that the test NLL for the cross-entropy model significantly increases towards the end of training (before plateauing), whereas the test NLL for the focal loss models remains low. To better understand this, we analyse the behaviour of these models

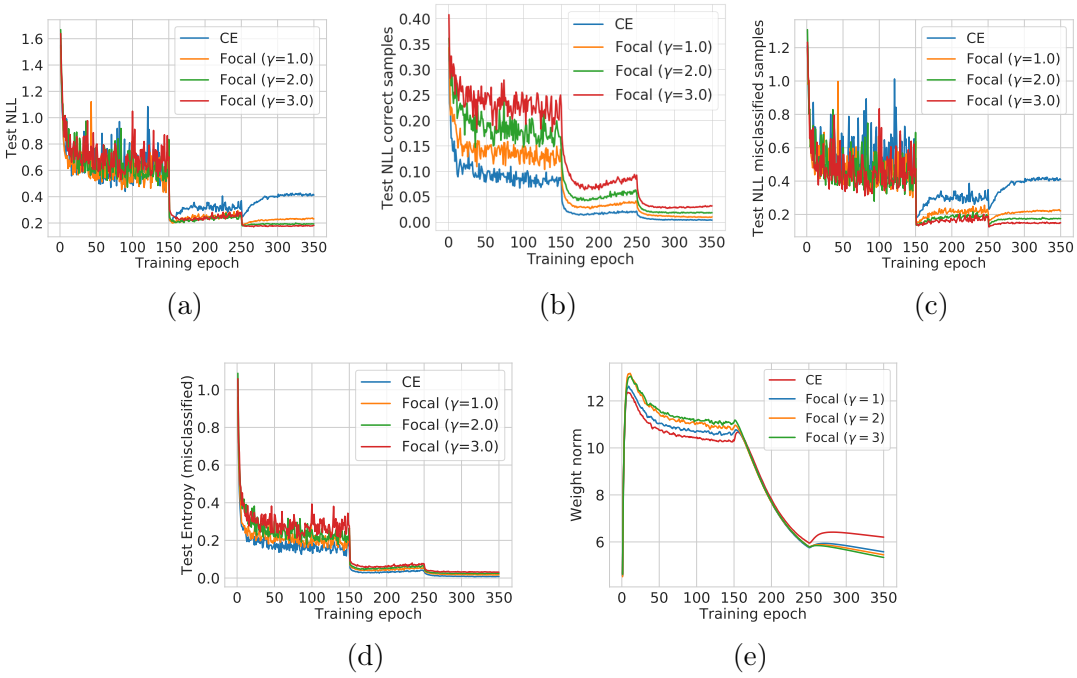


Figure 7.4: How metrics related to model calibration change whilst training several ResNet-50 networks on CIFAR-10, using either cross-entropy loss or focal loss with γ set to 1, 2 or 3.

separately for correctly and incorrectly classified samples. Figure 7.4(b) shows that even though the NLL for the correctly classified samples (mostly) decreases over the course of training for all models, the NLL for the focal loss models remain consistently higher than that for the cross-entropy model throughout training, implying that the focal loss models are relatively less confident than the cross-entropy model for samples that they do predict correctly. This is important, as we have already discussed that it is overconfidence that normally makes deep neural networks miscalibrated. Figure 7.4(c) shows that in contrast to the cross-entropy model, for which the NLL for misclassified test samples increases significantly after epoch 150, the rise in this value for the focal loss models is much less severe and almost absent for $\gamma = 3$. Additionally, in Figure 7.4(d), we notice that the entropy of the softmax distribution for misclassified test samples is consistently (if only marginally) higher for focal loss than for cross-entropy. This is consistent with Theorem 5.

Early Stopping Figure 7.4(a) suggests that applying early stopping when training a model on cross-entropy provides better calibration scores. However, we could not find an ideal way of doing early stopping that provides both the best calibration

Criterion	Loss	Epoch	Error	ECE %
ECE	NLL	151	7.34	1.69
ECE	Focal Loss	257	5.52	0.85
NLL	NLL	153	6.69	2.28
NLL	Focal Loss	266	5.34	1.33
Error	NLL	344	5.0	4.46
Error	Focal Loss	343	4.99	1.43
Full	NLL	350	4.95	4.35
Full	Focal Loss	350	4.98	1.55

Table 7.1: Classification errors and ECE scores obtained from ResNet-50 models trained using cross-entropy and focal loss with different early stopping criteria (best in hindsight ECE, NLL and classification error on the validation set) applied during training. In the table, the *Full* Criterion indicates models where early stopping has not been applied.

error and the best test-set accuracy. For a fair comparison, we trained ResNet50 networks on CIFAR-10 using both cross-entropy and focal loss with the best possible (in hindsight) early stopping. We trained each model for 350 epochs and chose the 3 intermediate models with the best validation set ECE, NLL, and classification error, respectively. We present the test-set performance in Table 7.1.

From the table, we can observe that:

1. For every early stopping criterion, focal loss outperforms cross-entropy in both test-set accuracy and ECE,

2. When using the validation set ECE as a stopping criterion, the intermediate model for cross-entropy indeed improves its test-set ECE, but at the cost of a significantly higher test error.

3. Even without early stopping, focal loss achieves consistently better error and ECE compared to cross-entropy using any stopping criterion.

As per Section 7.2, an increase in the test NLL and a decrease in the test entropy for misclassified samples, along with no corresponding increase in the test NLL for the correctly classified samples, can be interpreted as the network predicting softmax distributions for the misclassified samples that are ever more *peaky in the wrong place*. Notably, our results in Figures 7.4(b) to 7.4(d) clearly show that this effect is significantly reduced when training with focal loss rather than cross-entropy, leading to a better-calibrated network whose predictions are less peaky in the wrong place.

7.3.3 Theoretical justifications for focal loss

As mentioned previously, once a model trained using cross-entropy reaches high training accuracy, the learning algorithm tries to further reduce the training NLL by increasing the confidences of the correctly classified samples. It achieves this by magnifying the network weights to increase the magnitudes of the logits. To verify this hypothesis, we plot the L_2 norm of the weights of the last linear layer for all four networks from the previous section as a function of the training epoch in Figure 7.4(e). Notably, although the norms of the weights for the models trained on focal loss are initially higher than that for the cross-entropy model, *a complete reversal* in the ordering of the weight norms occurs between epochs 150 and 250. In other words, as the networks start to become miscalibrated, the weight norm for the cross-entropy model also starts to become greater than those for the focal loss models. This is because focal loss, by design, starts to regularise the network's weights once the model has gained a certain amount of confidence in its predictions. To better understand this, we consider the following Lemma

Theorem 6 (Relation between the gradients of cross-entropy and focal loss)**.** *For focal loss $\mathcal{L}_f$, with hyper-parameter γ, and cross-entropy loss $\mathcal{L}_c$ as defined before, the gradients with respect to the parameters for the last linear layer $\mathbf{w}$ can be related as*

$$\frac{\partial \mathcal{L}_{\mathrm{f}}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}_c}{\partial \mathbf{w}} g(\hat{p}_{i,y_i}, \gamma)$$

where $g(p, \gamma) = (1-p)^\gamma - \gamma p (1-p)^{\gamma-1} \log(p)$ and $\gamma \in \mathbb{R}^+$ is the focal loss hyperparameter. Thus,

$$\left\| \frac{\partial \mathcal{L}_{\mathrm{f}}}{\partial \mathbf{w}} \right\| \leq \left\| \frac{\partial \mathcal{L}_{\mathrm{c}}}{\partial \mathbf{w}} \right\|$$

if $g(\hat{p}_{i,y_i}, \gamma) \in [0, 1]$.

Proof in Appendix F.1

Theorem 6 shows the relationship between the norms of the gradients of the last linear layer for focal loss and cross-entropy loss, for the same network architecture. This relation depends on the function $g(p, \gamma)$, which we plot in Figure 7.5. It shows that for every γ, there exists a unique threshold p_0 such that for all $p \in [0, p_0]$,

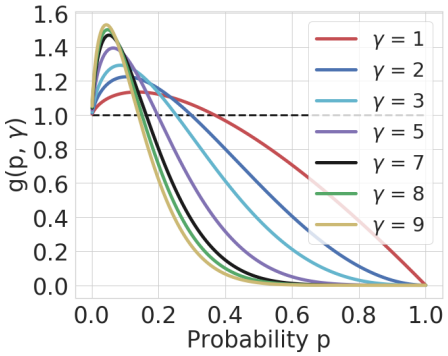
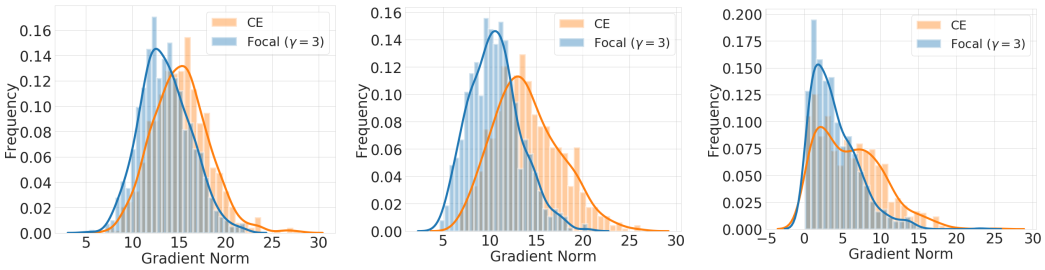


Figure 7.5: $g(p, \gamma)$ vs. p

$g(p, \gamma) \geq 1$, and for all $p \in (p_0, 1]$, $g(p, \gamma) < 1$. (For example, for $\gamma = 1$, $p_0 \approx 0.4$.) We use this insight to further explain why focal loss provides implicit weight regularisation.

Implicit weight regularisation: For a network trained using focal loss with a fixed γ, during the initial stages of the training, when $\hat{p}_{i,y_i} \in (0, p_0)$, $g(\hat{p}_{i,y_i}, \gamma) > 1$. Thus, the confidences of the focal loss model's predictions initially increase faster than cross-entropy. However, as soon as $\hat{p}_{i,y_i}$ crosses the threshold p_0, $g(\hat{p}_{i,y_i}, \gamma)$ falls below 1 and reduces the magnitude of the gradient updates made to the network's weights, thereby having a regularising effect on the weights. This is why, in Figure 7.4(e), we find that the norms of weights of the models trained with focal loss are initially higher than that for the model trained using cross-entropy. However, as training progresses, focal loss starts regularising the network weights and the ordering of the weight norms reverses. We can draw similar insights from Figures 7.6(a) to 7.6(c), in which we plot histograms of the gradient norms of the last linear layer (over all samples in the training set) at epochs 10, 100 and 200, respectively. At epoch 10, the gradient norms for cross-entropy and focal loss are similar, but as training progresses, those for cross-entropy decrease less rapidly than those for focal loss, indicating that the gradient norms for focal loss are consistently lower than those for cross-entropy throughout training.

Finally, observe in Figure 7.5 that for higher γ values, the fall in $g(p, \gamma)$ is steeper. We would thus expect a greater weight regularisation effect for models that use higher values of γ. This explains why, among the three models we trained using focal loss, the one with $\gamma = 3$ outperforms (in terms of calibration) the one with



(a) Epoch 10 (b) Epoch 100 (c) Epoch 200

Figure 7.6: Histograms of the gradient norms of the last linear layer for both cross-entropy and focal loss.

$\gamma = 2$, which in turn outperforms the model with $\gamma = 1$. Based on this observation, one might think that a higher value of gamma always leads to a more calibrated model. However, this is not the case, as we notice from Figure 7.5 that for $\gamma \geq 7$, $g(p, \gamma)$ reduces to nearly 0 for a relatively low value of p (around 0.5). As a result, using values of γ that are too high will cause the gradients to vanish (i.e. reduce to nearly 0) early, at a point at which the network's predictions remain ambiguous, thereby causing the training process to fail.

7.3.4 How to choose the focal loss hyper-parameter γ

We discussed that focal loss provides implicit entropy and weight regularisation and γ behaves akin to a regularisation coefficient. Lin et al. [165] fixed a γ, chosen by cross-validation, for all samples in the dataset. However, as we saw in Theorem 6, the regularisation effect for a sample i depends on $\hat{p}_{i,y_i}$, the predicted probability for the ground truth label for the sample. It thus makes sense to choose γ for each sample based on the value of $\hat{p}_{i,y_i}$. To this end, we provide Theorem 7.

Theorem 7 (Choosing the focal loss hyper-parameter)**.** *For a given $p_0 > 0$ and for all $1 \geq p \geq p_0$ and $\gamma \geq \gamma^* = \frac{a}{b} + \frac{1}{\log a} W_{-1}\big(-\frac{a^{(1-a/b)}}{b} \log a\big)$ where $a = 1 - p_0$, $b = p_0 \log p_0$, and W_{-1} is the Lambert-W function [58], the following holds*

$$g(p, \gamma) \leq 1$$

Moreover, for $p \geq p_0$ and $\gamma \geq \gamma^*$, the equality $g(p, \gamma) = 1$ holds only if $p = p_0$ and $\gamma = \gamma^*$.

Proof in Appendix F.1

It is worth noting that for all values of $p \geq p_0$ there exist multiple values of γ where $g(p, \gamma) \leq 1$. For a given p_0, Theorem 7 allows us to compute γ such that

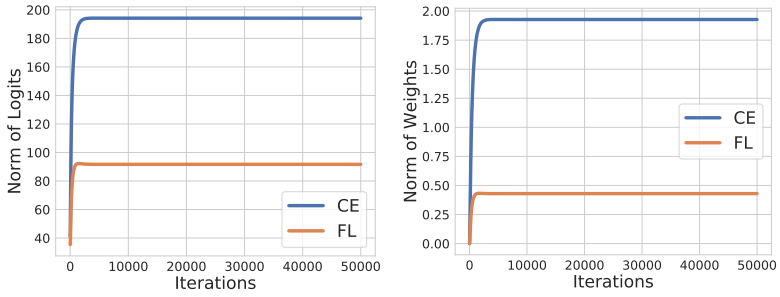
$$g(p, \gamma) = \begin{cases} 1 & p = p_0 \\ > 1 & p \in [0, p_0) \\ < 1 & p \in (p_0, 1] \end{cases}$$

This allows us to control the magnitude of the gradients for a particular sample i based on the current value of $\hat{p}_{i,y_i}$, and gives us a way of choosing a value of γ for each sample. For instance, a reasonable policy might be to choose γ s.t. $g(\hat{p}_{i,y_i}, \gamma) > 1$ if $\hat{p}_{i,y_i}$ is small (say less than 0.25), and $g(\hat{p}_{i,y_i}, \gamma) < 1$ otherwise. Such a policy will have the effect of making the weight updates larger for samples having a low predicted probability for the correct class and smaller for samples with a relatively higher predicted probability for the correct class.

Following the aforementioned arguments, we choose a threshold of $p_0 = 0.25$ and use Theorem 7 to obtain a policy for γ such that $g(p, \gamma)$ is observably greater than 1 for $p \in [0, 0.2)$ and $g(p, \gamma) < 1$ for $p \in (0.25, 1]$. In particular, we use the following schedule: if $\hat{p}_{i,y_i} \in [0, 0.25)$, then $\gamma = 5$, otherwise $\gamma = 3$ (note that $g(0.2, 5) \approx 1$ and $g(0.25, 3) \approx 1$: see Figure 7.5). We find this policy for γ to perform consistently well across multiple classification datasets and network architectures. Having said that, one can calculate multiple such schedules for γ following Proposition 7, using the intuition of having a relatively high γ for low values of $\hat{p}_{i,y_i}$ and a relatively low γ for high values of $\hat{p}_{i,y_i}$.

7.4 Training a linear model with focal loss and NLL

The behaviour of deep neural networks is generally quite different from linear models and the problem of calibration is more pronounced in the case of deep neural networks. Hence we focus on analysing the calibration of deep networks in this chapter. However, weight norm analysis for the various layers in a deep neural network is complex due to components of the training process like batchnorm and weight



(a) Norm of logits (b) Norm of weights.

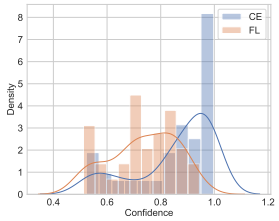
Figure 7.7: Norm of logits and weights of Linear Classifiers trained with Focal Loss (FL) and Cross-Entropy (CE)

decay. Hence, to see the effect of weight magnification on miscalibration, first, we use a generalised linear model on a simple data distribution.

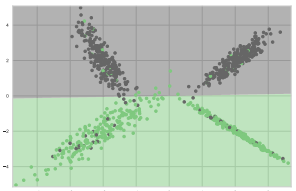
We consider a binary classification problem. The data matrix $\mathbf{X} \in \mathbb{R}^{2\times N}$ is created by assigning each class, two normally distributed clusters such that the means of the clusters are linearly separable. The means of the clusters are situated on the vertices of a two-dimensional hypercube of side length 4. The standard deviation for each cluster is 1 and the samples are randomly linearly combined within each cluster to add covariance. Further, for 10% of the data points, the labels are flipped. 4000 samples are used for training and 1000 samples are used for testing. The model consists of a simple 2-parameter logistic regression model. For a given $\mathbf{x} = (x_1, x_2)$, the model returns $f_{(w_1,w_2)}(\mathbf{x}) = \sigma(w_1x_1 + w_2x_2)$. We train this model using both cross-entropy and focal loss with $\gamma = 1$.

Weight magnification We have argued that focal loss implicitly regularises the weights of the model by providing smaller gradients as compared to cross-entropy. This helps in calibration as, if all the weights are large, the logits are large and thus the confidence of the network is large on all test points, even on the misclassified points. Consequently, when the model misclassifies, it misclassifies with high confidence. Figure 7.7 shows that the norms of the logits and the weights are much larger for the model trained with the cross-entropy loss as compared to the model trained with the focal loss.

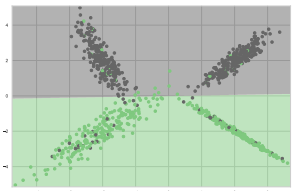
High confidence for mistakes Figures 7.8(b) and 7.8(c) show that gradient descent with cross-entropy (CE) and focal loss (FL) learns similar decision regions



(a) Confidence of misclassifications



(b) Decision boundary of linear classifier trained using cross-entropy



(c) Decision boundary of linear classifier trained using focal loss

Figure 7.8: Decision Boundary and confidences of Linear Classifiers.

i.e. the weight vector of the linear classifier points in the same direction. However, as we have seen that the norm of the weights is much larger for CE as compared to FL, we would expect the confidence of misclassified test points to be large for CE as compared to FL. A histogram of the confidence of the misclassified points, plotted in Figure 7.8(a) shows that CE almost always misclassifies with greater than 90% confidence whereas the confidence of misclassified samples is much lower for the FL model.

7.5 Experiments on real-world datasets

We conduct image and document classification experiments to test the performance of focal loss on more realistic models and datasets. For the former, we use CIFAR-10/100 [145] and Tiny-ImageNet [66] and train ResNet-50, ResNet-110 [110], Wide-ResNet-26-10 [289], and DenseNet-121 [119] models. For document classification experiments, we use the 20 Newsgroups [155] and Stanford Sentiment Treebank (SST) [242] datasets and train Global Pooling CNN [164] and Tree-LSTM [250] models. Further details on the datasets and training can be found in Appendix B.

Baselines Along with cross-entropy loss, we compare our method against the following baselines:

1. **MMCE** (Maximum Mean Calibration Error) [149], a continuous and differentiable proxy for calibration error that is used as a regulariser alongside cross-entropy,

2. **Brier loss** [37], the squared error between the predicted softmax vector and

Dataset	Model	Cross-Entropy		Brier Loss		MMCE		LS-0.05		FL-3 (Ours)		FLSD-53 (Ours)	
		Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T
CIFAR-100	ResNet-50	17.52	3.42(2.1)	6.52	3.64(1.1)	15.32	2.38(1.8)	7.81	4.01(1.1)	5.13	**1.97(1.1)**	**4.5**	2.0(1.1)
	ResNet-110	19.05	4.43(2.3)	**7.88**	4.65(1.2)	19.14	**3.86(2.3)**	11.02	5.89(1.1)	8.64	3.95(1.2)	8.56	4.12(1.2)
	Wide-ResNet-26-10	15.33	2.88(2.2)	4.31	2.7(1.1)	13.17	4.37(1.9)	4.84	4.84(1)	**2.13**	2.13(1)	3.03	**1.64(1.1)**
	DenseNet-121	20.98	4.27(2.3)	5.17	2.29(1.1)	19.13	3.06(2.1)	12.89	7.52(1.2)	4.15	**1.25(1.1)**	**3.73**	1.31(1.1)
CIFAR-10	ResNet-50	4.35	1.35(2.5)	1.82	1.08(1.1)	4.56	1.19(2.6)	2.96	1.67(0.9)	**1.48**	1.42(1.1)	1.55	**0.95(1.1)**
	ResNet-110	4.41	1.09(2.8)	2.56	1.25(1.2)	5.08	1.42(2.8)	2.09	2.09(1)	**1.55**	**1.02(1.1)**	1.87	1.07(1.1)
	Wide-ResNet-26-10	3.23	0.92(2.2)	**1.25**	1.25(1)	3.29	0.86(2.2)	4.26	1.84(0.8)	1.69	0.97(0.9)	1.56	**0.84(0.9)**
	DenseNet-121	4.52	1.31(2.4)	1.53	1.53(1)	5.1	1.61(2.5)	1.88	1.82(0.9)	1.32	1.26(0.9)	**1.22**	**1.22(1)**
Tiny-ImageNet	ResNet-50	15.32	5.48(1.4)	4.44	4.13(0.9)	13.01	5.55(1.3)	15.23	6.51(0.7)	1.87	1.87(1)	**1.76**	**1.76(1)**
20 Newsgroups	Global Pooling CNN	17.92	2.39(3.4)	13.58	3.22(2.3)	15.48	6.78(2.2)	**4.79**	2.54(1.1)	8.67	3.51(1.5)	6.92	**2.19(1.5)**
SST Binary	Tree-LSTM	7.37	2.62(1.8)	9.01	2.79(2.5)	5.03	4.02(1.5)	**4.84**	4.11(1.2)	16.05	1.78(0.5)	9.19	**1.83(0.7)**

Table 7.2: ECE (%) computed for different approaches both pre and post temperature scaling (cross-validating T on ECE). The optimal temperature for each method is indicated in brackets. $T \approx 1$ indicates an innately calibrated model.

the one-hot ground truth encoding, and

3. **Label smoothing** [189] (LS): given a one-hot target label distribution $\mathbf{q}$ and a smoothing factor α (hyperparameter), the smoothed vector $\mathbf{s}$ is obtained as $\mathbf{s}_i = (1-\alpha)\mathbf{q}_i + \alpha(1-\mathbf{q}_i)/(K-1)$, where $\mathbf{s}_i$ and $\mathbf{q}_i$ denote the i^{th} elements of $\mathbf{s}$ and $\mathbf{q}$ respectively, and K is the number of classes. Instead of $\mathbf{q}$, $\mathbf{s}$ is treated as the target label distribution during training. We train models using $\alpha = 0.05$ and $\alpha = 0.1$, but find $\alpha = 0.05$ to perform better. Thus, we report the results obtained from LS-0.05 with $\alpha = 0.05$.

Focal loss : We looked at different variants of focal loss, which varies in the way γ is assigned a value. When γ is fixed throughout training, we found $\gamma = 3$ to outperform $\gamma = 1$ and $\gamma = 2$. Thus, we use $\gamma = 3$ for our experiments and use the abbreviation FL-3 to report its experimental results.

We also tried multiple variants of the Sample-Dependant γ (c.f. Section 7.3.4) and found the variant which uses the following strategy to be most competitive — when $\hat{p}_{i,y_i} \in [0, 0.25)$, the value of γ is set to 5 and when $\hat{p}_{i,y_i} \in [0.25, 1)$, γ is set to 3. We call this approach Focal Loss (sample-dependent γ 5,3) and use the abbreviation FLSD-53 to report its experimental results.

Temperature scaling: To compute the optimal temperature for temperature scaling, we use two different methods: (a) learning the temperature by minimising validation set NLL and (b) performing grid search over temperatures between 0 and 10 with a step size of 0.1, and choosing the one that minimises validation set ECE. We find the second approach to produce *stronger baselines* and report all our

Dataset	Model	Cross-Entropy	Brier Loss	MMCE	LS-0.05	FL-3 (Ours)	FLSD-53 (Ours)
CIFAR-100	ResNet-50	23.3	23.39	23.2	23.43	22.75	23.22
	ResNet-110	22.73	25.1	23.07	23.43	22.92	22.51
	Wide-ResNet-26-10	20.7	20.59	20.73	21.19	19.69	20.11
	DenseNet-121	24.52	23.75	24.0	24.05	23.25	22.67
CIFAR-10	ResNet-50	4.95	5.0	4.99	5.29	5.25	4.98
	ResNet-110	4.89	5.48	5.4	5.52	5.08	5.42
	Wide-ResNet-26-10	3.86	4.08	3.91	4.2	4.13	4.01
	DenseNet-121	5.0	5.11	5.41	5.09	5.33	5.46
Tiny-ImageNet	ResNet-50	49.81	53.2	51.31	47.12	49.69	49.06
20 Newsgroups	Global Pooling CNN	26.68	27.06	27.23	26.03	29.26	27.98
SST Binary	Tree-LSTM	12.85	12.85	11.86	13.23	12.19	12.8

Table 7.3: Test set error (%) computed for different approaches.

results obtained using this approach.

7.5.1 Calibration and test accuracy

We report ECE% (computed using 15 bins) along with optimal temperatures in Table 7.2, and test-set error in Table 7.3. Firstly, for all dataset-network pairs, we obtain very competitive classification accuracies (shown in Table 7.3). This is important as it is easy to obtain a highly calibrated model while incurring a large test error by simply predicting a random class label with a uniform distribution over the classes. Secondly, *it is clear from Table 7.2 that focal loss with sample-dependent* γ *and with* $\gamma = 3$ *outperforms all the baselines: cross-entropy, label smoothing, Brier loss, and MMCE.* They produce the lowest calibration errors *both before and after temperature scaling.* This observation is particularly encouraging as it also indicates that a principled method for obtaining values of γ for focal loss can produce a very calibrated model with no need to use a validation set for tuning γ.

In Tables 7.4 to 7.6, we present the AdaECE, Classwise-ECE, and MCE scores for our models and compare it with all the baselines discussed above. The optimal temperature for each model is obtained by cross-validating it on ECE.

Finally, calibrated models should have a higher logit score (or softmax probability) on the correct class even when they misclassify, as compared to models which are less calibrated. Thus, intuitively, such models should have a higher Top-5 accuracy. Top-5 accuracy is the probability that one of the five classes with the top five conditional likelihoods is the correct class. In Table 7.7, we report the Top-5 accuracies for all our models on datasets where the number of classes is relatively high (i.e.,

Dataset	Model	Cross-Entropy		Brier Loss		MMCE		LS-0.05		FL-3 (Ours)		FLSD-53 (Ours)	
		Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T
CIFAR-100	ResNet-50	17.52	3.42(2.1)	6.52	3.64(1.1)	15.32	2.38(1.8)	7.81	4.01(1.1)	5.08	2.02(1.1)	**4.5**	**2.0(1.1)**
	ResNet-110	19.05	5.86(2.3)	**7.73**	4.53(1.2)	19.14	4.85(2.3)	11.12	8.59(1.1)	8.64	4.14(1.2)	8.55	**3.96(1.2)**
	Wide-ResNet-26-10	15.33	2.89(2.2)	4.22	2.81(1.1)	13.16	4.25(1.9)	5.1	5.1(1)	**2.08**	2.08(1)	2.75	**1.63(1.1)**
	DenseNet-121	20.98	5.09(2.3)	5.04	2.56(1.1)	19.13	3.07(2.1)	12.83	8.92(1.2)	4.15	**1.23(1.1)**	**3.55**	**1.24(1.1)**
CIFAR-10	ResNet-50	4.33	2.14(2.5)	1.74	**1.23(1.1)**	4.55	2.16(2.6)	3.89	2.92(0.9)	1.95	1.83(1.1)	**1.56**	1.26(1.1)
	ResNet-110	4.4	1.99(2.8)	2.6	1.7(1.2)	5.06	2.52(2.8)	4.44	4.44(1)	**1.62**	**1.44(1.1)**	2.07	1.67(1.1)
	Wide-ResNet-26-10	3.23	1.69(2.2)	1.7	1.7(1)	3.29	1.6(2.2)	4.27	2.44(0.8)	1.84	1.54(0.9)	**1.52**	**1.38(0.9)**
	DenseNet-121	4.51	2.13(2.4)	2.03	2.03(1)	5.1	2.29(2.5)	4.42	3.33(0.9)	**1.22**	1.48(0.9)	1.42	**1.42(1)**
Tiny-ImageNet	ResNet-50	15.23	5.41(1.4)	4.37	4.07(0.9)	13.0	5.56(1.3)	15.28	6.29(0.7)	1.88	1.88(1)	**1.42**	**1.42(1)**
20 Newsgroups	Global Pooling CNN	17.91	**2.23(3.4)**	13.57	3.11(2.3)	15.21	6.47(2.2)	**4.39**	2.63(1.1)	8.65	3.78(1.5)	6.92	2.35(1.5)
SST Binary	Tree-LSTM	7.27	3.39(1.8)	8.12	2.84(2.5)	**5.01**	4.32(1.5)	5.14	4.23(1.2)	16.01	2.16(0.5)	9.15	**1.92(0.7)**

Table 7.4: Adaptive ECE (%) computed for different approaches both pre and post temperature scaling (cross-validating T on ECE). Optimal temperature for each method is indicated in brackets.

Dataset	Model	Cross-Entropy		Brier Loss		MMCE		LS-0.05		FL-3 (Ours)		FLSD-53 (Ours)	
		Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T
CIFAR-100	ResNet-50	0.38	0.22(2.1)	0.22	0.20(1.1)	0.34	0.21(1.8)	0.23	0.21(1.1)	**0.20**	**0.20(1.1)**	**0.20**	**0.20(1.1)**
	ResNet-110	0.41	0.21(2.3)	0.24	0.23(1.2)	0.42	0.22(2.3)	0.26	0.22(1.1)	**0.24**	0.22(1.2)	**0.24**	**0.21(1.2)**
	Wide-ResNet-26-10	0.34	0.20(2.2)	0.19	0.19(1.1)	0.31	0.20(1.9)	0.21	0.21(1)	**0.18**	**0.18(1)**	**0.18**	0.19(1.1)
	DenseNet-121	0.45	0.23(2.3)	0.20	0.21(1.1)	0.42	0.24(2.1)	0.29	0.24(1.2)	0.20	0.20(1.1)	**0.19**	**0.20(1.1)**
CIFAR-10	ResNet-50	0.91	0.45(2.5)	0.46	0.42(1.1)	0.94	0.52(2.6)	0.71	0.51(0.9)	0.43	0.48(1.1)	**0.42**	**0.42(1.1)**
	ResNet-110	0.91	0.50(2.8)	0.59	0.50(1.2)	1.04	0.55(2.8)	0.66	0.66(1)	**0.44**	**0.41(1.1)**	0.48	0.44(1.1)
	Wide-ResNet-26-10	0.68	0.37(2.2)	0.44	0.44(1)	0.70	0.35(2.2)	0.80	0.45(0.8)	0.44	0.36(0.9)	**0.41**	**0.31(0.9)**
	DenseNet-121	0.92	0.47(2.4)	0.46	0.46(1)	1.04	0.57(2.5)	0.60	0.50(0.9)	0.43	0.41(0.9)	**0.41**	**0.41(1)**
Tiny-ImageNet	ResNet-50	0.22	0.16(1.4)	0.16	0.16(0.9)	0.21	0.16(1.3)	0.21	0.17(0.7)	0.16	0.16(1)	**0.16**	**0.16(1)**
20 Newsgroups	Global Pooling CNN	1.95	0.83(3.4)	1.56	**0.82(2.3)**	1.77	1.10(2.2)	**0.93**	0.91(1.1)	1.31	1.05(1.5)	1.40	1.19(1.5)
SST Binary	Tree-LSTM	5.81	3.76(1.8)	6.38	2.48(2.5)	**3.82**	**2.70(1.5)**	3.99	3.20(1.2)	6.35	2.81(0.5)	4.84	3.24(0.7)

Table 7.5: Classwise-ECE (%) computed for different approaches both pre and post temperature scaling (cross-validating T on ECE). Optimal temperature for each method is indicated in brackets.

on CIFAR-100 with 100 classes and Tiny-ImageNet with 200 classes). We observe focal loss with sample-dependent γ to produce the highest top-5 accuracies on all models trained on CIFAR-100 and the second-best top-5 accuracy (only marginally below the highest accuracy) on Tiny-ImageNet.

7.5.2 Confident and calibrated models

It is worth noting that focal loss with sample-dependent γ has optimal temperatures that are very close to 1, mostly lying between 0.9 and 1.1 (see Table 7.2). This property is shown by the Brier loss and label smoothing models as well, albeit with worse calibration errors. By contrast, the temperatures for cross-entropy and MMCE models are significantly higher, with values lying between 2.0 and 2.8. An optimal temperature close to 1 indicates that the model is innately calibrated and cannot be made significantly more calibrated by temperature scaling. In fact, a temperature much greater than 1 can make a model underconfident in general as it is applied to all predictions irrespective of the correctness of the model's outputs.

We follow the approach adopted in Kumar et al. [149] and measure the percentage

Dataset	Model	Cross-Entropy		Brier Loss		MMCE		LS-0.05		FL-3 (Ours)		FLSD-53 (Ours)	
		Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T	Pre T	Post T
CIFAR-100	ResNet-50	44.34	12.75(2.1)	36.75	21.61(1.1)	39.53	11.99(1.8)	26.11	18.58(1.1)	**13.02**	**6.76(1.1)**	16.12	27.18(1.1)
	ResNet-110	55.92	22.65(2.3)	24.85	12.56(1.2)	50.69	19.23(2.3)	36.23	30.46(1.1)	26	13.06(1.2)	**22.57**	**10.94(1.2)**
	Wide-ResNet-26-10	49.36	14.18(2.2)	14.68	13.42(1.1)	40.13	16.5(1.9)	23.79	23.79.1(1)	**9.96**	9.96(1)	10.17	**9.73(1.1)**
	DenseNet-121	56.28	21.63(2.3)	15.47	8.55(1.1)	49.97	13.02(2.1)	43.59	29.95(1.2)	11.61	6.17(1.1)	**9.68**	**5.68(1.1)**
CIFAR-10	ResNet-50	38.65	20.6(2.5)	31.54	22.46(1.1)	60.06	23.6(2.6)	35.61	40.51(0.9)	21.83	**15.76(1.1)**	**14.89**	26.37(1.1)
	ResNet-110	44.25	29.98(2.8)	25.18	22.73(1.2)	67.52	31.87(2.8)	45.72	45.72(1)	25.15	37.610(1.1)	**18.95**	**17.35(1.1)**
	Wide-ResNet-26-10	48.17	26.63(2.2)	77.15	77.15(1)	36.82	32.33(2.2)	24.89	37.53(0.8)	**23.86**	**25.64(0.9)**	74.07	36.56(0.9)
	DenseNet-121	45.19	32.52(2.4)	19.39	19.39(1)	43.92	27.03(2.5)	45.5	53.57(0.9)	77.08	76.27(0.9)	**13.36**	**13.36(1)**
Tiny-ImageNet	ResNet-50	30.83	13.33(1.4)	8.41	12.82(0.9)	26.48	12.52(1.3)	25.48	17.2(0.7)	6.11	6.11(1)	**3.76**	**3.76(1)**
20 Newsgroups	Global Pooling CNN	36.91	36.91(3.4)	31.35	31.35(2.3)	34.72	34.72(2.2)	**8.93**	**8.93(1.1)**	18.85	18.85(1.5)	17.44	17.44(1.5)
SST Binary	Tree-LSTM	71.08	88.48(1.8)	92.62	91.86(2.5)	68.43	32.92(1.5)	39.39	**35.72(1.2)**	**22.32**	74.52(0.5)	73.7	76.71(0.7)

Table 7.6: MCE (%) computed for different approaches both pre and post temperature scaling (cross-validating T on ECE). Optimal temperature for each method is indicated in brackets.

Dataset	Model	Cross-Entropy		Brier Loss		MMCE		LS-0.05		FLSD-53 (Ours)	
		Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
CIFAR-100	ResNet-50	76.7	93.77	76.61	93.24	76.8	93.69	76.57	92.86	76.78	**94.44**
	ResNet-110	77.27	93.79	74.9	92.44	76.93	93.78	76.57	92.27	77.49	**94.78**
	Wide-ResNet-26-10	79.3	93.96	79.41	94.56	79.27	94.11	78.81	93.18	79.89	**95.2**
	DenseNet-121	75.48	91.33	76.25	92.76	76	91.96	75.95	89.51	77.33	**94.49**
Tiny-ImageNet	ResNet-50	50.19	74.24	46.8	70.34	48.69	73.52	52.88	**76.15**	50.94	76.07

Table 7.7: Top-1 and Top-5 accuracies for different approaches.

Dataset	Model	Cross-Entropy (Pre T)		Cross-Entropy (Post T)		MMCE (Pre T)		MMCE (Post T)		Focal Loss (Pre T)		Focal Loss (Post T)	
		|S99|%	Accuracy	|S99|%	Accuracy	|S99|%	Accuracy	|S99|%	Accuracy	|S99|%	Accuracy	|S99|%	Accuracy
CIFAR-10	ResNet-110	97.11	96.33	11.5	97.39	97.65	96.72	10.62	99.83	61.41	99.51	31.10	99.68
CIFAR-10	ResNet-50	95.93	96.72	7.33	99.73	92.33	98.24	4.21	100	46.31	99.57	14.27	99.93

Table 7.8: Percentage of test samples predicted with confidence higher than 99% and the corresponding accuracy for Cross Entropy, MMCE and Focal loss computed both pre and post temperature scaling (represented in the table as pre T and post T respectively).

of test samples that are predicted with a confidence of 0.99 or more (we call this set of test samples $S99$). In Table 7.8, we report $|S99|$ as a percentage of the total number of test samples, along with the accuracy of the samples in $S99$ for ResNet-50 and ResNet-110 trained on CIFAR-10, using cross-entropy loss, MMCE loss, and focal loss. We observe that $|S99|$ for the focal loss model is much lower than for the cross-entropy or MMCE models before temperature scaling. However, after temperature scaling, $|S99|$ for focal loss is significantly higher than for both MMCE and cross-entropy. The reason is that with an optimal temperature of 1.1, the confidence of the temperature-scaled model for focal loss does not reduce as much as it does for models trained with cross-entropy and MMCE, for which the optimal temperatures lie between 2.5 and 2.8. We thus conclude that models trained on focal loss are not only more calibrated, but also preserve their confidence on predictions, even after being post-processed with temperature scaling.

8. Accelerating Encrypted Prediction via Binary Neural Networks

In this chapter, we will look at an issue that is commonly faced when neural networks are deployed in the real world in a *Prediction As a Service* framework (see Section 3.4 for a discussion of this framework). In particular, we look at a setting where a service provider has trained a machine learning model and users can use that model by sending their private data to the service provider. However, neither the user wants the service provider to be able to read their private data or the output of the model on that data nor does the service provider want the user to learn anything about their machine learning model. We propose some strict computational and privacy requirements that encapsulate this setting and refer to this paradigm as *Encrypted Prediction As A Service* (EPAAS).

One way of implementing this is through the technique of Fully Homomorphic Encryption (FHE), which allows for arbitrary computations on encrypted data without decrypting it. However, this can be incredibly slow in practice rendering the technique useless in the real world. In this chapter, along with proposing the EPAAS framework, we also discuss ways to make the implementation of FHE for EPAAS faster in practice so that it can be applied on modern complex neural networks.

8.1 Main contributions

In this chapter, our focus is on achieving speed-ups when using complex models with fully homomorphically encrypted data. To achieve these speed-ups, we propose several methods to modify the training and design of neural networks, as well as algorithmic tricks to parallelise and accelerate computation on encrypted data. In particular, we propose the following techniques

- We propose the use of binary neural networks to make EPAAS faster without sacrificing much on its test accuracy.
- We propose two types of circuits for performing inner products between un-

encrypted and encrypted data: reduce tree circuits and sorting networks. We give a runtime comparison of each method.

- We introduce an easy trick, which we call the *+1 trick* to sparsify encrypted computations.

- We demonstrate that our techniques are easily parallelisable and we report timing for a variety of computation settings on real-world datasets, alongside classification accuracies.

Most similar to our work is Bourse et al. [33] who use neural networks with signed integer weights and binary activations to perform encrypted predictions. However, their method is only evaluated on MNIST and achieves accuracy comparable to a linear classifier (92%) [156], and the encryption scheme parameters depend on the structure of the model, potentially requiring clients to re-encrypt their data if the service provider updates their model. Our framework allows the service provider to update their model at any time and allows one to use binary neural networks of Courbariaux et al. [61] which, in particular, achieve high accuracy on MNIST (99.04%). Another closely related work is Meehan et al. [175] who design encrypted adder and multiplier circuits so that they can implement machine learning models on integers. This can be seen as complementary to our work on binary networks: while they achieve improved accuracy because of greater precision, they are less efficient than our methods (on MNIST we achieve the same accuracy with a 29× speedup, via our sparsification and parallelisation tricks).

Private training. In this thesis, we do not address the question of training machine learning models with encrypted data. There has been some recent work in this area [107, 13]. However, as of now, it appears possible only to train very small models using fully homomorphic encryption. We leave this for future work.

8.2 Encrypted prediction as a Service

In this section, we describe our *Encrypted Prediction as a Service (EPAAS)* paradigm. We then detail our privacy and computational guarantees. Finally, we discuss how different related work is suited to this paradigm and propose a solution.

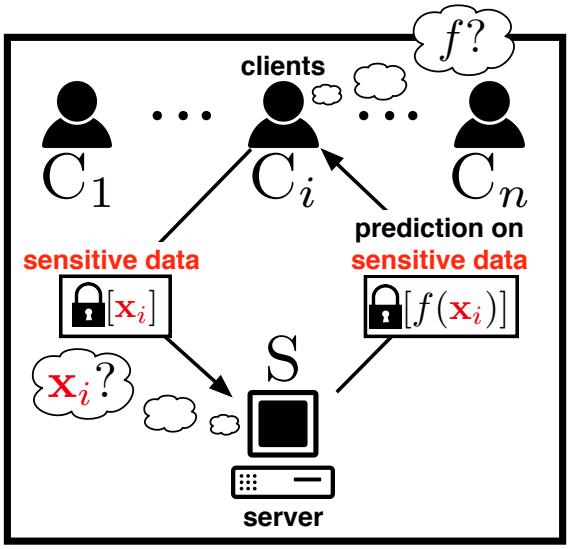


Figure 8.1: Encrypted prediction as a service.

	Privacy		**Computation**			
Prior Work	P1	P2	C1	C2	C3(i)	C3(ii)
CryptoNets [88]	✓	-	✓	✓	-	✓
[45]	✓	-	✓	✓	-	✓
[33]	✓	✓	✓	✓	-	✓
MPC [184, 166] [218, 46, 135]	✓	-	✓	-	-	-
[175], Ours	✓	✓	✓	✓	✓	✓

Table 8.1: Privacy and computational guarantees of existing methods for sensitive data classification.

In the EPAAS setting we have any number of clients, say $C_1, \ldots, C_n$ that have data $\mathbf{x}_1, \ldots, \mathbf{x}_n$. The clients would like to use a highly accurate model f provided by a server S to predict some outcome. In cases where data $\mathbf{x}$ is not sensitive, there are already many solutions for this such as BigML[1], Wise.io[2], Google Cloud AI[3], Amazon Machine Learning[4], among others. However, if the data is sensitive so that the clients would be uncomfortable giving the raw data to the server, none of these systems can offer the client a prediction.

[1]https://wise.io/

[2]https://cloud.google.com/products/machine-learning/

[3]https://bigml.com/

[4]https://aws.amazon.com/machine-learning/

8.2.1 Privacy and computational guarantees

If data $\mathbf{x}$ is sensitive (e.g., $\mathbf{x}$ may be the health record of client C, and $f(\mathbf{x})$ may be the likelihood of heart disease), then we would like to have the following privacy guarantees:

P1. Neither the server S nor any other party, learn anything about client data $\mathbf{x}$, other than its size *(privacy of the data)*.

P2. Neither the client C nor any other party, learn anything about model f, other than the prediction $f(x)$ given client data $\mathbf{x}$ (and whatever can be deduced from it) *(privacy of the model)*.

Further, the main attraction of EPAAS is that the client is involved as little as possible. More concretely, we wish to have the following computational guarantees:

C1. No external party is involved in the computation.

C2. The rounds of communication between client and server should be limited to 2 (send data & receive prediction).

C3. Communication and computation at the client-side should be independent of model f. In particular, (i) the server should be able to update f without communicating with any client, and (ii) clients should not need to be online during the computation of $f(\mathbf{x})$.

Note that these requirements rule out protocols with preprocessing stages or that involve third parties. Generally speaking, a satisfactory solution based on FHE would proceed as follows: (1) a client generates encryption parameters, encrypts their data $\mathbf{x}$ using the private key, and sends the resulting encryption $\tilde{\mathbf{x}}$, as well as the public key to the server. (2) The server evaluates f on $\tilde{\mathbf{x}}$ leveraging the homomorphic properties of the encryption, to obtain an encryption $\tilde{f}(\mathbf{x})$ without learning anything whatsoever about $\mathbf{x}$, and sends $\tilde{f}(\mathbf{x})$ to the client. (3) Finally, the client decrypts and recovers the prediction $f(\mathbf{x})$ in the clear. A high-level depiction of these steps is shown in Figure 8.1.

8.2.2 Existing approaches

Table 8.1 describes whether prior work satisfies the above privacy and computational guarantees. First, note that Cryptonets [88] violates C3(i) and P2. This is because the clients would have to generate parameters for the encryption according to the structure of f, so the client can make inferences about the model (violating P2) and the client is not allowed to change the model f without telling the client (violating C3(i)). The same holds for the work of Chabanne et al. [45]. The approach of Bourse et al. [33] requires the server to calibrate the parameters of the encryption scheme according to the magnitude of intermediate values, thus C3(i) is not necessarily satisfied. Closely related to our work is that of Meehan et al. [175] which satisfies our privacy and computational requirements. We will show that our method is significantly faster than this method, with very little sacrifice in accuracy.

Multi-Party Computation (MPC). It is important to distinguish between approaches based purely on homomorphic encryption (described above), and those involving Multi-Party Computation (MPC) techniques, such as [184, 166, 223, 218, 46, 135]. While generally, MPC approaches are faster, they crucially rely on all parties being involved in the whole computation, which conflicts with requirement C3(ii). Additionally, in MPC the structure of the computation is public to both parties, which means that the server would have to communicate basic information such as the number of layers of f. This conflicts with requirements P2, C2, and C3(i).

In this work, we propose to use a very tailored homomorphic encryption technique to guarantee all privacy and computational requirements. In the next section, we give background on homomorphic encryption. Further, we motivate the encryption protocol and the machine learning model class we use to satisfy all guarantees.

8.3 Background

All cryptosystems define two functions: 1. an encryption function $\mathcal{E}(\cdot)$ that maps data (often called *plaintexts*) to encrypted data (*ciphertexts*); 2. a decryption function $\mathcal{D}(\cdot)$ that maps ciphertexts back to plaintexts. In public-key cryptosystems, to

evaluate the encryption function $\mathcal{E}$, one needs to hold a public key k_{PUB}, so the encryption of data x is $\mathcal{E}(x, k_{\text{PUB}})$. Similarly, to compute the decryption function $\mathcal{D}(\cdot)$ one needs to hold a secret key k_{SEC} which allows us to recover: $\mathcal{D}(\mathcal{E}(x, k_{\text{PUB}}), k_{\text{SEC}}) = x$.

A cryptosystem is *homomorphic* in some operation $\blacksquare$ if it is possible to perform another (possibly different) operation $\square$ such that: $\mathcal{E}(x, k_{\text{PUB}}) \square \mathcal{E}(x, k_{\text{PUB}}) = \mathcal{E}(x \blacksquare y, k_{\text{PUB}})$. Finally, in this work we assume all data to be binary $\in \{0, 1\}$. For more detailed background on FHE beyond what is described below, see the excellent tutorial of Halevi [101].

8.3.1 Fully Homomorphic Encryption

In 1978, cryptographers posed the question: *Does an encryption scheme exist that allows one to perform arbitrary computations on encrypted data?* The implications of this, called a *Fully Homomorphic Encryption* (FHE) scheme, would enable clients to send computations to the cloud while retaining control over the secrecy of their data. This was still an open problem however 30 years later. Then, in 2009, a cryptosystem [84] was devised that could, in principle, perform such computations on encrypted data. Similar to previous approaches, in each computation, noise is introduced into the encrypted data. And after a certain number of computations, the noise grows too large so that the encryptions can no longer be decrypted. The key innovation was a technique called *bootstrapping*, which allows one to reduce the noise to its original level without decrypting.

At a high level, the idea is as follows. Assume the cryptosystem can evaluate a version of its decryption function where the secret key is also encrypted: $\mathcal{D}(\mathcal{E}(x, k_{\text{PUB}}), \mathcal{E}(k_{\text{SEC}}, k_{\text{PUB}}))$. Gentry [84] showed that if the secret key was newly encrypted then the output of this function would be an encrypted version of data x *with all of the noise removed.* Finally, also assume this cryptosystem can homomorphically evaluate a `NAND` gate on encrypted data. Then because via `NAND` one can express all possible logical operations, such a cryptosystem would be fully homomorphic. Gentry [84] subsequently derived such a scheme, the first FHE scheme. That result constituted a massive breakthrough, as it established, for the first time,

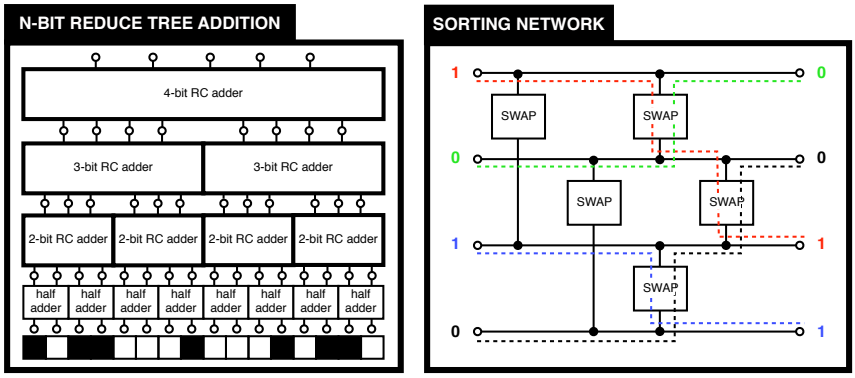


Figure 8.2: Binary circuits used for inner product: reduce tree (*Left*) and sorting network (*Right*). RC is short for ripple-carry.

a fully homomorphic encryption scheme [84]. Unfortunately, the original bootstrapping procedure was highly impractical, as massive noise was introduced in each homomorphic operation. Consequently, much of the research since the first FHE scheme has been devoted to reducing the growth of noise so that the scheme never has to perform bootstrapping. Indeed, even in recent FHE schemes bootstrapping is slow (roughly six minutes in a highly optimised implementation of a recent popular scheme [102]) and bootstrapping many times increases the memory requirements of encrypted data.

Encrypted prediction with levelled HE

Thus, one common technique to implement encrypted prediction was to take an existing ML algorithm and approximate it with as few operations as possible, to never have to bootstrap. This involved careful parameter tuning to ensure that the security of the encryption scheme was sufficient, that it didn't require too much memory, and that it ran in a reasonable amount of time. One prominent example of this is Cryptonets [88]. If a practitioner wanted to add a few layers to the neural network model, they would need to ensure that all of the operations could still be performed without bootstrapping. Otherwise, security parameters or previous layers would need to be adjusted to account for the added layers.

Encrypted prediction with FHE

Recent developments in cryptography call for rethinking this approach. Ducas and Micciancio [72] devised a scheme that that could bootstrap a single Boolean gate in

under one second with reduced memory. Recently, Chillotti et al. [53] introduced optimisations implemented in the TFHE library, which further reduced bootstrapping to under 0.1 seconds. In this chapter, we demonstrate that this change has a huge impact on designing encrypted machine learning algorithms. Specifically, encrypted computation is now modular: the cost of adding a few layers to an encrypted neural network is simply the added cost of each layer in isolation. This is particularly important as recent developments in deep learning such as Residual Networks [110] and Dense Networks [120] have shown that networks with many layers are crucial to achieving state-of-the-art accuracy.

8.3.2 Binary Neural Networks

The cryptosystem that we will use in this chapter, TFHE, is however restricted to computing binary operations. We note that, concurrent to the work that led to TFHE, was the development of neural network models that perform binary operations between binary weights and binary activations. These models, called Binary Neural Networks (BNNs), were first devised by Kim and Smaragdis [139] and Courbariaux et al. [61], and were motivated by the prospect of training and testing deep models on limited memory and limited compute devices, such as mobile phones.

Technical details. We now describe the technical details of binary networks that we will aim to replicate on encrypted data. In a *Binary Neural Network* (BNN) every layer maps a binary input $\mathbf{x} \in \{-1, 1\}^d$ to a binary output $\mathbf{z} \in \{-1, 1\}^p$ using a set of binary weights $\mathbf{W} \in \{-1, 1\}^{(p,d)}$ and a binary activation function $\texttt{sign}(\cdot)$ that is 1 if $x \geq 0$ and -1 otherwise. Although binary nets do not typically use a bias term, applying batch-normalization [125] when evaluating the model means that a bias term $\mathbf{b} \in \mathbb{Z}^p$ may need to be added before applying the activation function (cf. Sec. 8.4.1). Thus, when evaluating the model, a fully connected layer in a BNN implements the following transformation $\mathbf{z} := \text{sign}(\mathbf{W}\mathbf{x} + \mathbf{b})$. From now on we will call all data represented as $\{-1, 1\}$ *non-standard binary* and data represented as $\{0, 1\}$ as *binary*. Kim and Smaragdis [139], Courbariaux et al. [61] were the first to note that the above inner product nonlinearity in BNNs could be implemented using the following steps:

1. Transform data and weights from non-standard binary to binary: $\mathbf{w}, \mathbf{x} \to \overline{\mathbf{w}}, \overline{\mathbf{x}}$ by replacing -1 with 0. Now all data is in $\{0, 1\}$.

2. Apply element-wise multiplication by using the logical `XNOR` operator$(\overline{\mathbf{w}}, \overline{\mathbf{x}})$ for each element of $\overline{\mathbf{w}}$ and $\overline{\mathbf{x}}$.

3. Sum the result of the previous step by using `popcount` operation (which counts the number of 1s), call this S.

4. If the bias term is b, check if $2S \geq d - b$, if so the activation is positive and return 1, otherwise return -1.

Thus we have that,

$$z_i = \text{sign}\left(2 \cdot \text{popcount}(\text{XNOR}(\overline{\mathbf{w}}_i, \overline{\mathbf{x}})) - d + b\right)$$

Related binary models. Since the initial work on BNNs there has been a wealth of work on binarising, ternarising, and quantizing neural networks Chen et al. [49], Courbariaux et al. [60], Han et al. [104], Hubara et al. [123], Zhu et al. [298], Chabanne et al. [45], Chen et al. [50]. Our approach is currently tailored to methods that have binary activations and we leave the implementation of these methods on encrypted data for future work.

8.4 Methods

In this work, we observe that BNNs can be run on encrypted data by designing circuits in TFHE for computing their operations. We consider boolean circuits that operate on encrypted data and unencrypted weights and biases. We show how these circuits allow us to efficiently implement the three main layers of binary neural networks: fully connected, convolutional, and batch-normalization. We then show how a simple trick that allows us to sparsify our computations. Our techniques can be easily parallelised. During the evaluation of a circuit, gates at the same level in the tree representation of the circuit can be evaluated in parallel. Hence, when implementing a function, "shallow" circuits are preferred in terms of parallelisation. While

parallel computation was often used to justify employing the second generation FHE techniques—where parallelisation comes from ciphertext packing—we show in the following section that our techniques create dramatic speedups for a state-of-the-art FHE technique. We emphasise that a key challenge is that we need to use *data oblivious* algorithms (circuits) when dealing with encrypted data as the algorithm never discovers the actual value of any query made on the data.

8.4.1 Binary OPs

The three primary circuits we need are for the following tasks: 1. computing the inner product; 2. computing the binary activation function (described in the previous section) and; 3. dealing with the bias.

Encrypted inner product

As described in the previous section, BNNs can speed up an inner product by computing XNORs (for element-wise multiplication) followed by a POPCOUNT (for summing). In our case, we compute an inner product of size d by computing XNORs element-wise between d bits of encrypted data and d bits of unencrypted data, which results in an encrypted d bit output. To sum this output, the POPCOUNT operation is useful when weights and data are unencrypted because POPCOUNT is implemented in the instruction set of Intel and AMD processors, but when dealing with encrypted data we simply resort to using shallow circuits. We consider two circuits for summation, both with sublinear depth: a reduce tree adder and a sorting network.

Reduce tree adder. We implement the sum using a binary tree of half and ripple-carry (RC) adders organised into a reduction tree, as shown in Figure 8.2 (*Left*). All these structures can be implemented to run on encrypted data because TFHE allows us to compute XNOR, AND, and OR on encrypted data. The final number returned by the reduction tree $\tilde{S}$ is the binary representation of the number of 1s resulting from the XNOR, just like POPCOUNT. Thus, to compute the BNN activation function $\text{sign}(\cdot)$ we need to check whether $2\tilde{S} \geq d - b$, where d is the number of bits in $\tilde{S}$ and b is the bias. Note that if the bias is zero we simply need to check if $\tilde{S} \geq d/2$. To do

so we can simply return the second-to-last bit of $\tilde{S}$. If it is 1 then $\tilde{S}$ is at least $d/2$. If the bias b is non-zero (because of batch-normalization, described in Section 8.4.1), we can implement a circuit to perform the check $2\tilde{S} \geq d - b$. The bias b (which is available in the clear) may be an integer as large as $\tilde{S}$. Let $\mathbb{B}[(d-b)/2]$ and $\mathbb{B}[\tilde{S}]$ be the binary representations of $(d-b)/2$ and $\tilde{S}$ respectively. Algorithm 7 describes a comparator circuit that returns an encrypted value of 1 if the above condition holds and (encrypted) 0 otherwise (where $\text{MUX}(s, a, b)$ returns a if $s = 1$ and b otherwise). As encrypted operations dominate the running time of our computation, in practice this computation essentially corresponds to evaluating d MUX gates. This gate has a dedicated implementation in TFHE, which results in a very efficient comparator in our setting.

Algorithm 7 Comparator

Inputs: Encrypted $\mathbb{B}[\tilde{S}]$, unencrypted $\mathbb{B}[(d-b)/2]$, size d of $\mathbb{B}[(d-b)/2]$,$\mathbb{B}[\tilde{S}]$
Output: Result of $2\tilde{S} \geq d - b$

1: $o = 0$
2: **for** $i = 1, \ldots, d$ **do**
3: **if** $\mathbb{B}[(d-b)/2]_i = 0$ **then**
4: $o = \text{MUX}(\mathbb{B}[\tilde{S}]_i, \tilde{1}, o)$
5: **else**
6: $o = \text{MUX}(\mathbb{B}[\tilde{S}]_i, o, \tilde{0})$
7: **end if**
8: **end for**
9: **Return:** o

Sorting network. We do not technically care about the sum of the result of the element-wise XNOR between $\bar{\mathbf{w}}$ and $\bar{\mathbf{x}}$. In fact, all we care about is if the result of the comparison: $2\tilde{S} \geq d - b$. Thus, another idea is to take the output of the (bitwise) XNOR and sort it. Although this sorting needs to be performed over encrypted data, the rest of the computation does not require any homomorphic operations; after sorting we hold a sequence of encrypted 1s, followed by encrypted 0s. To output the correct value, we only need to select one of the (encrypted) bit in the correct position and return it. If $b = 0$ we can simply return the encryption of the central bit in the sequence; indeed, if the central bit is 1, then there are more 1s than 0s and thus $2\tilde{S} \geq d$ and we return 1. If $b \neq 0$ we need to offset the returned index by b in the correct direction depending on the sign of b. To sort the initial

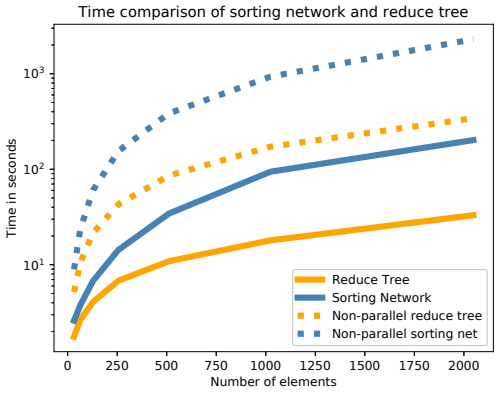


Figure 8.3: Timing of sorting network and reduce tree addition for different sized vectors, with and without parallelisation.

array we implement a sorting network, shown in Figure 8.2 (*Right*). The sorting network is a sequence of swap gates between individuals bits, where $\text{SWAP}(a, b) = (\text{OR}(a, b), \text{AND}(a, b))$. Note that if $a \geq b$ then $\text{SWAP}(a, b) = (a, b)$, and otherwise is (b, a). More specifically, we implement Batcher's sorting network [24], which consists of $O(n \log^2(n))$ swap gates, and has depth $O(\log^2(n))$.

Batch normalization

Batch normalization is mainly used during training; however, during the evaluation of a model, this requires us to scale and translate and scale the input (which is the output of the previous layer). In practice, when our activation function is the sign $(\cdot)$ function, this only means that we need to update the bias term (the actual change to the bias term is an elementary calculation). As our circuits are designed to work with a bias term, and the scaling and translation factors are available as plaintext (as they are part of the model), this operation is easily implemented during test time.

8.4.2 Sparsification via "+1"-trick

Since we have access to the weight matrix $\mathbf{W} \in \{-1, 1\}^{p \times d}$ and the bias term $\mathbf{b} \in \mathbb{Z}^p$ in the clear (only data $\mathbf{x}$ and subsequent activations are encrypted), we can exploit the fact that $\mathbf{W}$ always has values ± 1 to, roughly, halve the cost computation. We

consider $\mathbf{w} \in \{-1, 1\}^d$ which is a single row of $\mathbf{W}$ and observe that:

$$\mathbf{w}^\top \mathbf{x} = (\mathbf{1} + \mathbf{w})^\top (\mathbf{1} + \mathbf{x}) - \sum_i w_i - (\mathbf{1} + \mathbf{x})^\top \mathbf{1},$$

where $\mathbf{1}$ denotes the vector in which every entry is 1. Further note that $(\mathbf{1} + \mathbf{w}) \in \{0, 2\}^d$ which means that the product $(\mathbf{1} + \mathbf{w})^\top (\mathbf{1} + \mathbf{x})$ is simply the quantity $4 \sum_{i:w_i=1} \bar{x}_i$, where $\bar{\mathbf{x}}$ refers to the standard binary representation of the non-standard binary $\mathbf{x}$. Assuming at most half of the w_is were originally $+1$, if $w \in \{-1, 1\}^d$, only $d/2$ encrypted values need to be added. We also need to compute the encrypted sum $\sum_i x_i$; however, this latter sum need only be computed once, no matter how many output units the layer has. Thus, this small bit of extra overhead roughly *halves* the amount of computation required. We note that if $\mathbf{w}$ has more -1s than $+1$s, $\mathbf{w}^\top \mathbf{x}$ can be computed using $(\mathbf{1} - \mathbf{w})$ and $(\mathbf{1} - \mathbf{x})$ instead. This guarantees that we never need to sum more than half the inputs for any output unit. The sums of encrypted binary values can be calculated as described in Sec. 8.4.1. The overheads are two additions required to compute $(\mathbf{1} + \mathbf{x})^\top \mathbf{1}$ and $(\mathbf{1} - \mathbf{x})^\top \mathbf{1}$, and then a subtraction of two $\log(d)$-bit long encrypted numbers. (The multiplication by 2 or 4 as may be sometimes required is essentially free, as bit shifts correspond to dropping bits, and hence do not require homomorphic operations). As our experimental results show this simple trick roughly halves the computation time of one layer; the actual savings appear to be even more than half as in many instances the number of elements we need to sum over is significantly smaller than half.

It is worth emphasizing the advantage of binarising and then using the above approach to making the sums sparse. By default, units in a neural network compute an affine function to which an activation function is subsequently applied. The affine map involves an inner product that involves d multiplications. Multiplication under fully homomorphic encryption schemes is, however, significantly more expensive than addition. By binarising and applying the above calculation, we've replaced the inner product operation by selection (which is done in the clear as $\mathbf{W}$ is available in plaintext) and (encrypted) addition.

8.4.3 Ternarisation (Weight Dropping)

Ternary neural networks use weights in $\{-1, 0, 1\}$ rather than $\{-1, 1\}$; this can alternatively be viewed as dropping connections from a BNN. Using ternary neural networks rather than binary reduces the computation time as encrypted inputs for which the corresponding w_i is 0 can be safely dropped from the computation, before the method explained in section 8.4.2 is applied to the remaining elements. Our experimental results show that a binary network can be ternarised to maintain the same level of test accuracy with roughly a quarter of the weights being 0 (cf. Sec. 8.5.3).

8.5 Experimental Results

In this section, we report encrypted binary neural network prediction experiments on several real-world datasets. We begin by comparing the efficiency of the two circuits used for inner product, the reduce tree and the sorting network. We then describe the datasets and the architecture of the BNNs used for classification. We report the classification timings of these BNNs for each dataset, for different computational settings. Finally, we give accuracies of the BNNs compared to floating-point networks. Our code is freely available at [251].

8.5.1 Reduce tree vs. sorting network

We show timings of reduce tree and sorting network for a different number of input bits, with and without parallelisation in Figure 8.3 (parallelisation is over 16 CPUs). We notice that the reduce tree is strictly better when comparing parallel or non-parallel timings of the circuits. As such, from now on we use the reduce tree circuit for inner product.

It should be mentioned that at the outset this result was not obvious because while sorting networks have more levels of computation, they have fewer gates. Specifically, the sorting network used for encrypted sorting is the bitonic sorting network which for n bits has $O(\log^2 n)$ levels of computation whereas the reduce tree only has $O(\log n)$ levels. On the other hand, the reduce tree requires 2 gates for

each half adder and $5k$ gates for each k-bit RC adder, whereas a sorting network only requires 2 gates per SWAP operation. Another factor that may slow down sorting networks is that our implementation of sorting networks is recursive, whereas the reduce tree is iterative.

Datasets We evaluate on four datasets, three of which have privacy implications due to health care information (datasets Cancer and Diabetes) or applications in surveillance (dataset Faces). We also evaluate on the standard benchmark MNIST dataset. All of the datasets and the corresponding model architectures are described in Appendix B.

8.5.2 Timing

We give timing results for the classification of an instance in different computational settings. All of the strategies use the parallel implementations of the reduce tree circuit computed across 16 CPUs (the solid line orange line in Figure 8.3). The *Out Seq* strategy computes each operation of a BNN sequentially (using the parallel reduce tree circuit). Notice that for any layer of a BNN mapping d inputs to p output nodes, the computation over each of the p output nodes can be parallelised. From this, we can easily estimate the effect of further parallelisation over the outputs of BNN layers, as the encrypted computation will remain identical. The *Out 16-P* strategy estimates parallelizing the computation of the p output nodes across a cluster of 16 machines (each with 16 CPUs). The *Out Full-P* strategy estimates complete parallelisation, in which each output node can be computed independently on a separate machine. We note that for companies that already offer prediction as a service, both of these parallel strategies are not unreasonable requirements. Indeed it is not uncommon for such companies to run hundreds of CPUs/GPUs over multiple days to tune hyperparameters for deep learning models[1]. Additionally, we report how timings change with the introduction of the *+1-trick* is described in Section 8.4.2. These timings are given in Table 8.2 (computed with Intel Xeon CPUs @ 2.40GHz, processor number E5-2673V3). We notice that without parallelisation over BNN outputs, the predictions on datasets that use fully connected layers: Cancer

[1]https://tinyurl.com/yc8d79oe

Parallelism	Cancer	Diabetes	Faces	MNIST
Out Seq	3.5s	283s	763.5h	65.1h
+1-trick	3.5s	250s	564h	37.22 h
Out 16-P. +1 trick	3.5s	31.5 s	33.1h	2.41 h
Out Full-P	3.5s	29s	1.3h	147s

Table 8.2: Neural Network timings on various datasets using different forms of parallelism.

and Diabetes, finish within seconds or minutes. While for the datasets that use convolutional layers: Faces and MNIST, predictions require multiple days. The *+1-trick* cuts the time of MNIST prediction by half and reduces the time of Faces prediction by 200 hours. With only a bit of parallelism over outputs (*Out 16-Parallel*) prediction on the Faces dataset now requires less than 1.5 days and MNIST can be done in 2 hours. With complete parallelism (*Out N-Parallel*) all methods reduce to under 2 hours.

8.5.3 Accuracy

We wanted to ensure that BNNs can still achieve similar test-set accuracies to floating-point networks. To do so, for each dataset we construct similar floating-point networks. For the Cancer dataset, we use the same network except we use the original 30 real-valued features, so the fully connected layer is $30 \rightarrow 1$, as was used in Meehan et al. [175]. For Diabetes and Faces, just like for our BNNs we cross-validate to find the best networks (for Faces: 4 convolutional layers, with filter sizes of 5×5 and 64 output channels; for Diabetes the best network is the same as used in the BNN). For MNIST we report the accuracy of the best performing method as reported in Wan et al. [267][2]. Additionally, we report the accuracy of the weight-dropping method described in Section 8.4. The results are shown in Table 8.3. We notice that apart from the Faces dataset, the difference in accuracies between the floating-point networks and BNNs are at most 1.2% (on MNIST). The face dataset uses a different network in floating-point which seems to be able to exploit the increased precision to increase accuracy by 5.1%. We also observe that

[2]https://tinyurl.com/knn2434

	Cancer	**Diabetes**	**Faces**	**MNIST**
Floating	0.977	0.556	0.942	0.998
BNN	0.971	0.549	0.891	0.986
BNN drop 10%	0.976	0.549	0.879	0.976
BNN drop 20%	0.912	0.541	0.878	0.973

Table 8.3: The accuracy of floating-point networks compared with BNNs, with and without weight dropping. The Cancer dataset floating-point accuracy is given by [175], the MNIST floating-point accuracy is given by [267], and the MNIST BNN accuracy (without dropping) is given by [61].

weight dropping by 10% reduces the accuracy by at most 1.2% (on Faces). Dropping 20% of the weights seem to have a small effect on all datasets except Cancer, which has only a single layer and so likely relies more on every individual weight.

Epilogue

This thesis looks at four different concepts of reliability in the context of deep neural networks — generalisation, adversarial robustness, calibration, and privacy. As various existing works [292, 249, 99, 88], as well as this thesis, point out, these problems seem to have arisen due to deep neural networks getting larger, more *complex*, and also, more accurate. The ability of modern deep neural networks to memorise noise makes it difficult to certify their generalisability (Chapter 4) and it provably makes them more vulnerable to adversarial attacks (Chapter 5). The large capacity of these networks, which allows them to memorise label noise, also makes them prone to NLL-overfitting (Chapter 7) and to miscalibration. Further, as we see in Chapter 8, the network architectures of these deep neural networks are ill-suited to satisfy the EPAAS constraints. Thus, the same large capacity which allows them to obtain high test accuracy also results in these side-effects, hurting their reliability.

This poses a deeper question as to whether these quantities of reliability are fundamentally at odds with accuracy. This question has been addressed theoretically in several works, some of which we discuss in Chapter 3, especially for adversarial robustness. Some works have suggested that to get a low adversarial error, we need more data [234, 185] or more computation [64, 39]. In Chapter 5, we show theoretical examples where we do not need either of these; choosing the inductive biases carefully suffices for robust generalisation. Chapter 6 shows that a low rank prior over the representation space (using the LR-layer) achieves a small adversarial error without suffering in test accuracy.

We show similar results for the other notions of reliability. In particular, we show that by using proper regularisations and network architectures, we can obtain less memorisation of label noise (SRN in Chapter 4), more calibrated models (Focal loss in Chapter 7), and privacy guarantees for user data (BNNs in Chapter 8) without sacrificing accuracy or needing more data or computation. Interestingly, both SRN and LR-layer regularises variants of the rank operator. While SRN controls a soft

version of the rank of the parameters, the LR-layer controls the hard rank of the representations. While existing works mostly look at norm-based regularisations, our work suggests that a rank-like measure, which controls the effective degree of freedom, is more effective.

All of these works show that choosing the right inductive biases, by way of regularisations and network architectures, has a significant impact on obtaining a simultaneously reliable and accurate model, with limited data and computation. However, we have not completely minimised the apparent tradeoff yet. For example, the adversarial robustness of our models in Chapter 6 is not as high as the models trained with AT (which albeit has a lower accuracy). Assuming that this tradeoff should not exist as we, humans, do not suffer from it, it is an open question to determine whether we need more data, more computation, better architectures, or perhaps all of these to completely mitigate the tradeoff.

Requiring more data or computation is often considered undesirable, but we should perhaps recall the progress that has been made, along this direction, in the last decade, and be hopeful that this will be a much lesser constraint in the coming years with the rapid advances in computer architecture and crowdsourcing services. Finally, I would like to end by remarking that as we develop new and more powerful neural network architectures and learning algorithms, we should be benchmarking them not just on traditional metrics of performance like accuracy, but also on metrics of reliability such as those discussed in this thesis.

A. Mathematical prerequisites

This Appendix will define some mathematical preliminaries that are used in the thesis and recall some things we already know in Linear Algebra, Statistics and Probability. While most of the things in this appendix is fairly basic, it has been included for the sake of completeness and to avoid looking up other resources.

Basic Linear Algebra

We start by assuming that the reader has a basic knowledge of vector spaces and linear operators on the elements of vector spaces. As this is not the main topic of discussion, we only skim over topics that are indispensable to the understanding of this document. For a broader introduction to the basics of Linear Algebra, the reader is directed to Strang [246].

Specifically, a vector is an element of a vector space and a matrix is a linear operator on vectors[1] i.e. it transfers vectors from one vector space to another. In most cases, we will deal with the vector space $\mathbb{R}^d$ where d. Wherever the dimension of the matrix (or the domain of the linear transformation) is not mentioned, it should be clear from the context.

Eigenvalues and Eigenvectors If $\mathbf{W}$ is a square matrix and $\mathbf{x}$ is a non-zero vector, such that the vector $\mathbf{Wx}$ is a scalar multiple of $\mathbf{x}$, i.e. $\mathbf{Wx} = \lambda\mathbf{x}$, then $\mathbf{x}$ is an eigenvector of $\mathbf{A}$ and λ is the corresponding eigenvalue. However, eigenvectors and eigenvalues only exist for square matrices. Singular Value Decomposition or SVD is a decomposition for any general rectangular matrix $\mathbf{A}$, $\text{SVD}(\mathbf{A}) = \mathbf{U\Sigma V}^\top$ where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices (i.e. $\mathbf{U}^\top\mathbf{U} = I$) and $\mathbf{\Sigma}$ is a non-negative diagonal matrix. Its diagonal entries are usually arranged in descending order and are known as singular values. We use $\lambda_i(\mathbf{A})$ and $\sigma_i(\mathbf{A})$ to denote the i^{th} largest eigenvalue and singular value of $\mathbf{A}$ respectively. Specifically, $\lambda_{\max}(\mathbf{A})$ and $\lambda_{\min}(\mathbf{A})$ represents the largest and smallest eigenvalues of $\mathbf{A}$, and $\sigma_{\max}(\mathbf{A})$ and $\sigma_{\min}(\mathbf{A})$ denotes the largest and smallest singular values of $\mathbf{A}$.

[1] Only true for finite vector spaces but we will avoid the mathematical complexities for simplicity.

Vector norms For a vector $\mathbf{x} \in \mathbb{R}^d$, $\|\mathbf{x}\|_p$ represents its ℓ_p norm, defined as $\|\mathbf{x}\|_p = \sqrt[p]{\sum_{i=1}^{d} x_i^p}$ where $\mathbf{x}_i$ indexes the i^{th} element of $\mathbf{x}$. By default, $\|\mathbf{x}\|$ represents the ℓ_2 norm of $\mathbf{x}$. $\|\cdot\|_0$ is not strictly a norm but it often referred to as the ℓ_0 norm and measures the sparsity of its argument vector.

Matrix operator norms The most common matrix norms are the induced operator norms. If $\|\cdot\|_\alpha$ is a vector norm then the induced operator norm is defined as

$$\|\mathbf{A}\|_\alpha = \sup_{x \neq \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_\alpha}{\|\mathbf{x}\|_\alpha} \tag{A.1}$$

Perhaps, the most common operator norm is the ℓ_2 operator norm, also known as the spectral norm and is induced by $\|\cdot\|_2$. In Chapter 4, we discuss lipschitzness using the α, β-operator norm which is defined as

$$\|\mathbf{A}\|_{\alpha,\beta}^{\text{op}} = \sup_{x \neq \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_\beta}{\|\mathbf{x}\|_\alpha} \tag{A.2}$$

The notation with op in the super-script is necessary to differentiate it from the entry-wise p, q matrix norm defined below.

ℓ_p-schatten norms The ℓ_p schatten norm is applied on a matrix by applying the corresponding ℓ_p norm on the vector of singular values of the matrix. We do not use this norm in this thesis to avoid confusion with other matrix norms. However, it is interesting to note that the ℓ_2 operator norm defined above is equivalent to the ℓ_∞-schatten norm. The ℓ_2-schatten norm is the ℓ_2 norm of the singular values and is also known as the Frobenius norm ($\|\cdot\|_{\text{F}}$ of the matrix. The rank of a matrix, which is equal to the number of non-zero singular values can be thought of as the ℓ_0-schatten norm though neither the rank nor the ℓ_0 *norm* is a true norm.

Entry-wise matrix norms Finally, the last kind of matrix norms we look at are the entry-wise norms. The ℓ_p entry-wise norms are applied on a matrix by unrolling the matrix as one long vector and applying the norm on that vector. The most common entry-wise norms for matrices is the ℓ_2 entry-wise norm, also known as the

Frobenius norm, and is denoted as $\|\cdot\|_{\mathrm{F}}$. The Frobenius norm of a matrix $\mathbf{A}$ is

$$\|\mathbf{A}\|_{\mathrm{F}} = \sqrt{\sum_i \sum_j \mathbf{A}_{i,j}^2} \tag{A.3}$$

where $\mathbf{A}_{i,j}$ represents the $(i,j)^{th}$ entry of the matrix $\mathbf{A}$. Equation (3.6), uses a slightly different variant of the entry-wise norm known as the p,q entry-wise norm

$$\|\mathbf{A}\|_{p,q} = \sqrt[q]{\sum_{j=1}^{n} \left(\sqrt[p]{\sum_{i=1}^{m} (\mathbf{A}_{i,j})^p} \right)^q} \tag{A.4}$$

To summarize, we use the following norms frequently in this thesis. The spectral norm of a matrix $\mathbf{A}$ is represented by both $\|\mathbf{A}\|$ and $\|\mathbf{A}\|_2$ (its operator norm notation). $\|\mathbf{A}\|_F$ represents the Frobenius norm of a matrix $\mathbf{A}$. $\mathrm{rank}\,(\mathbf{A})$ denotes the rank of $\mathbf{A}$ i.e. the number of non-zero singular values of $\mathbf{A}$ or the ℓ_0 schatten norm of A[2]. The p,q operator norm is denoted as $\|\cdot\|_{p,q}^{\mathrm{op}}$ and the p,q entry-wise norm is denoted as $\|\cdot\|_{p,q}$.

Positive semi-definiteness of matrices An important kind of matrix that often appears in learning are matrices $\mathbf{M}$ such that $\forall x,\ x^\top M x \,\&0$ where $\& \in \{<, \leq, >, \geq\}$, Such a matrix is called negative definite, negative semi-definite, positive definite and positive semi-definite respectively. Clearly, a matrix with all positive eigenvalues is positive definite (PD), a matrix with all non-negative eigenvalues is positive semi-definite (PSD) and similarly for negative definite (ND) and negative semi-definite matrix (NSD).

A commonly used PSD matrix is the outer-product matrix. For any vector $\mathbf{z} \in \mathbb{R}^{d\times 1}$, the outer product matrix $\mathbf{z}\mathbf{z}^\top$ is PSD. This can be shown as follows

$$\forall \mathbf{x} \in \mathbb{R}^{d\times 1}, \quad \mathbf{x}^\top \mathbf{z}\mathbf{z}^\top \mathbf{x} = \left\langle \mathbf{z}^\top \mathbf{x}, \mathbf{z}^\top, \mathbf{x} \right\rangle = \left\| \mathbf{z}^\top \mathbf{x} \right\|_2^2 \geq 0 \tag{A.5}$$

[2] ℓ_0 norms are not true norms. Mathematically, this is a misuse of notation.

Common Inequalities

The following inequalities are used in various parts of the thesis and are listed here for ease of reference.

Inequality 1. *For any* $x \in \mathbb{R}$,

$$1 + x \leq e^x$$

While this can be easily proved through differentiation, a proof without using differentiation is presented below.

Proof. We know that e^x can be expanded as follows

$$e^x = 1 + x + \sum_{k=2}^{\infty} \frac{x^k}{k!}$$

We will look at the following three cases.

- $x \leq -1$ We know e^x is always positive and in this case $1 + x$ is non-positive. Hence, the inequality trivially holds.
- $x \geq 0$ By the expansion above, we can write $e^x = 1 + x + p$, where p is some non-negative term. Hence, the inequality trivially holds.
- $x \in (-1, 0)$ Rewriting the expansion as follows completes the proof.

$$e^x - (1 + x) = \sum_{k=1}^{\infty} \frac{x^{2k}}{2k!} + \frac{x^{2k+1}}{(2k+1)!} = \sum_{k=1}^{\infty} \frac{x^{2k}}{(2k+1)!} (2k + 1 - |x|) \geq 0$$

□

Inequality 2 (Union bound)**.** *Consider any finite or countable collection of events denoted as* $(A_1, A_2, A_3 \ldots,)$*, then the probability of occurring of at least one of the events from that collection is no more than the sum of the individual probabilities of all the events in that collection i.e.*

$$\mathbb{P}\left[\bigcup_i A_i\right] \leq \sum_i \mathbb{P}\left[A_i\right]$$

Inequality 3 (Jensen's Inequality)**.** *For a convex real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ and an integrable function g defined on $[a, b]$, such that f is defined on, at least, the co-domain of g, the following holds —*

$$f\left(\frac{1}{b-a}\int_a^b g\right) \leq \frac{1}{b-a}\int_a^b g\,(f)$$

Inequality 4 (Holder's Inequality and the Cauchy Schwartz Inequality)**.** *Let $\mathbf{f}, \mathbf{g}$ be real-valued vectors and consider any $p, q \in [1, \infty)$ such that $\frac{1}{p} + \frac{1}{q} = 1$, then*

$$|\langle \mathbf{f}, \mathbf{g} \rangle| \leq \|\mathbf{f}\|_p \|\mathbf{g}\|_q$$

For $p = q = 2$, this reduces to the Cauchy-Schwartz inequality

$$|\langle \mathbf{f}, \mathbf{g} \rangle| \leq \|\mathbf{f}\| \|\mathbf{g}\|$$

Inequality 5 (Markov's Inequality)**.** *For a non-negative random variable X with its mean $\mu = \mathbb{E}[X]$ and a positive number a,*

$$\mathbb{P}[X \geq a] \leq \frac{\mu}{a}$$

Inequality 6 (Chernoff Bound)**.** *Suppose $X_1 \cdots X_m$ are m $\{0, 1\}$-valued independent random with $\mathbb{P}[X_i = 1] = p_i$ where $0 \leq p_i \leq 1$. Denote $X = \sum_{i=1}^m X_i$ and $\mu = \mathbb{E}[X] = \sum_{i=1}^m p_i$. Then $\forall \delta,\ 0 \leq \delta \leq 1$ the Chernoff bound states that*

$$\mathbb{P}[X \geq (1+\delta)\mu] \leq \exp\left(-\frac{\delta^2 \mu}{(3)}\right)$$

$$\mathbb{P}[X \leq (1-\delta)\mu] \leq \exp\left[-\frac{\delta^2 \mu}{2}\right]$$

Combining them together, we can write

$$\mathbb{P}[|X - \mu| \geq \delta\mu] \leq 2\exp\left(-\frac{\delta^2 \mu}{3}\right)$$

B. Common datasets and neural network architectures

B.1 Datasets

We use the following image and document classification datasets in our experiments:

1. **MNIST** [156]: This dataset has $60,000$ training and $10,000$ test images. Each image has a dimension of 28x28. This dataset is used in Chapters 5 and 8. For experiments in Chapter 8, we use the model (torch7) described in [61] whereas the model used in Chapter 5 is described in the chapter itself.

2. **CIFAR-10** [145]: This dataset has 60,000 colour images of size 32×32, divided equally into 10 classes. For all experiments other than the ones in Chapter 7, we use the standard train/test split of $50,000/10,000$ images. In Chapter 7, we use a train/validation/test split of $45,000/5,000/10,000$ images. Furthermore, we augment the training images by applying random crops and random horizontal flips. This dataset is used in the experiments for Chapters 4 to 7

3. **CIFAR-100** [145]: This dataset has 60,000 colour images of size 32×32, divided equally into 100 fine classes. There is also a fixed partitioning of the 100 classes into 20 coarse classes. Note that the images in this dataset are not the same images as in CIFAR-10. We also augment the training images by applying random crops and random horizontal flips. We again use a train/validation/test split of 45,000/5,000/10,000 images. For all experiments other than the ones in Chapter 7, we again use the standard train/test split of $50,000/10,000$ images. In Chapter 7, we use a train/validation/test split of $45,000/5,000/10,000$ images. This dataset is used in the experiments for Chapters 4 to 7

4. **SVHN** [193]: The Street View House Number or SVHN dataset has $73,257$ digits for training, $26,032$ digits for testing, and $531,131$ additional easier samples, to be used as extra training data. There are 10 classes like MNIST and CIFAR10. We use this dataset in Chapter 6.

5. **Restricted Imagenet Settings** [259] Restricted-Imagenet is a subset of ImageNet with 64 x 64 dimensional images. There are 60 fine classes and 10 coarse classes with each coarse class having 6 distinct fine classes in them. The train set size is 77237 and the test-set size is 3000. The fine classes within each coarse are balanced i.e. given a coarse class all the fine classes in it are equally represented in this dataset. This dataset is used in the experiments for Chapter 5.

Coarse Class	Fine Classes
Dog	Chihuahua, Japanese spaniel, Maltese dog, Pekinese, Shih-Tzu,Blenheim spaniel
Bird	cock, hen, ostrich, brambling, goldfinch, house finch
Insect	tiger beetle,ladybug,ground beetle, long-horned beetle, leaf beetle, dung beetle
Monkey	guenon, patas, baboon, macaque, langur, colobus
Car	jeep, limousine,cab, beach wagon, ambulance, convertible
Feline	leopard, snow leopard, jaguar, lion, cougar, lynx
Truck	tow truck, moving van, fire engine, garbage truck, pickup,police van
Fruit	Granny Smith, rapeseed, corn, acorn, hip, buckeye
Fungus	gyromitra, hen-of-the-woods, coral fungus,stinkhorn, agaric, earthstar
Boat	gondola, fireboat, speedboat, lifeboat, yawl, canoe

Table B.1: Fine-grained classes in Restricted Imagenet

6. **Tiny-ImageNet** [66]: Tiny-ImageNet is a subset of ImageNet with 64 x 64 dimensional images, 200 classes and 500 images per class in the training set and 50 images per class in the validation set. The image dimensions of Tiny-ImageNet are twice that of CIFAR-10/100 images.

   For Tiny-ImageNet, we train for 100 epochs with a learning rate of 0.1 for the first 40 epochs, 0.01 for the next 20 epochs and 0.001 for the last 40 epochs. We use a training batch size of 64. It should be noted that for Tiny-ImageNet, we saved 50 samples per class (i.e., a total of 10000 samples) from the training set as our own validation set to fine-tune the temperature parameter (hence, we trained on 90000 images) and we use the Tiny-ImageNet validation set as our test set. This dataset is used in Chapter 7.

7. **CelebA** [168] The CelebA dataset contains $202,599$ coloured images scaled to a size of 64×64. We use this dataset for the experiments on image generation in Chapter 4.

8. **20 Newsgroups** [155]: This dataset contains 20,000 news articles, categorised evenly into 20 different newsgroups based on their content. It is a popular dataset for text classification. Whilst some of the newsgroups are very related (e.g. rec.motorcycles and rec.autos), others are quite unrelated (e.g. sci.space and misc.forsale). We use a train/validation/test split of 15,098/900/3,999 documents.

   On this dataset, we train the Global Pooling Convolutional Network [164] using the Adam optimiser, with learning rate 0.001, and betas 0.9 and 0.999.[1]. We used Glove word embeddings [208] to train the network. We trained all the models for 50 epochs and used the models with the best validation accuracy. This dataset is used in Chapter 7.

9. **Stanford Sentiment Treebank (SST)** [242]: This dataset contains movie reviews in the form of sentence parse trees, where each node is annotated by sentiment. We use the dataset version with binary labels, for which 6,920/872/1,821 documents are used as the training/validation/test split. In the training set, each node of a parse tree is annotated as positive, neutral or negative. At test time, the evaluation is done based on the model classification at the root node, i.e. considering the whole sentence, which contains only positive or negative sentiment.

   On this dataset, we train the Tree-LSTM [250] using the AdaGrad optimiser with a learning rate of 0.05 and a weight decay of 10^{-4}, as suggested by the authors. We used the constituency model, which considers binary parse trees of the data and trains a binary Tree-LSTM on them. The Glove word embeddings [208] were also tuned for best results. The code framework we used is inspired by [258]. We trained these models for 25 epochs and used the models with the best validation accuracy. This dataset is used in Chapter 7.

[1]The code is a PyTorch adaptation of `https://github.com/aviralkumar2907/MMCE`

10. **Cancer** [71]The Cancer dataset[2] contains 569 data points where each point has 30 real-valued features. The task is to predict whether a tumor is malignant (cancerous) or benign. We use this dataset in Chapter 8. Similar to Meehan et al. [175] we divide the dataset into a training set and a test in a $70:30$ ratio. For every real-valued feature, we divide the range of the feature into three equal-spaced bins and one-hot encode each feature by its bin-membership. This creates a 90-dimensional binary vector for each example. We use a single fully connected layer $90 \to 1$ followed by a batch normalization layer, as is common practice for BNNs [61].

    Diabetes [245]This dataset[3] contains data on $100,000$ patients with diabetes. The task is to predict one of three possible labels regarding hospital readmission after release: 1. a patient is readmitted to the hospital after release in less than or equal to 30 days; 2. readmission happens after 30 days;3. a patient is not readmitted. We use this dataset in Chapter 8. We divide patients into a 80/20 train/test split. We ensure that the same patient does not appear in both the training and test dataset by splitting patients who appear more than once by putting them in either the training or the test dataset in a $80:20$ split ratio. As this dataset contains real and categorical features, we bin them as in the Cancer dataset. We obtain a $1,704$ dimensional binary data point for each entry. Our network (selected by cross validation) consists of a fully connected layer $1704 \to 10$, a batch normalization layer, a SIGN activation function, followed by another fully connected layer $10 \to 3$, and a batch normalization layer.

11. **Labeled Faces in the Wild-a** [121, 275] The *Labeled Faces in the Wild-a* dataset contains $13,233$ gray-scale face images. We use the binary classification task of gender identification from the images. We use this dataset in Chapter 8. We use the *LFW-a* version [4] of the dataset where the images are aligned using a commercial alignment software and are grayscale. We resize the images to size 50×50. Our network architecture (selected by cross-validation) contains 5 con-

[2]https://tinyurl.com/gl3yhzb
[3]https://tinyurl.com/m6upj7y
[4]https://www.openu.ac.il/home/hassner/data/lfwa/

volutional layers, each of which is followed by a batch normalization layer and a SIGN activation function (except the last which has no activation). All convolutional layers have unit stride and filter dimensions 10×10. All layers except the last layer have 32 output channels (the last has a single output channel). The output is flattened and passed through a fully connected layer $25 \to 1$ and a batch normalization layer.

For all our experiments, we use the PyTorch framework, setting any hyperparameters not explicitly mentioned to the default values used in the standard models.

B.2 Neural network architectures

We use the following neural network architectures throughout the thesis.

1. **ResNet-18/50/110** The ResNet-18/50/100 are standard $18, 50$, and 110 layered ResNets respectively with batch Norm and ReLU. Each of these networks have a convolution layer followed by four residual blocks and a final fully connected layer. The depth of these blocks vary depending on the total number of layers.

   When using these networks to learn CIFAR10 and CIFAR100, we train with stochastic gradient descent for a total of 350 epochs with an initial learning rate of 0.1, which is multiplied 0.1 after 150 and 250 epochs respectively, a weight decay of $5e-4$ and a momentum of 0.9. We use ResNet-18 in Chapters 5 and 6, ResNet-50 in Chapters 5 to 7, and ResNet-100 in Chapters 4 and 7

2. **WideResNet-28/26-10** We use a standard WideResNet with 28 and 26 layers and a growth factor of 10. In total, the network has 36,539,124 trainable parameters. The network is the standard configuration with batchnorm and ReLU activations and is trained with a weight decay of $1e-4$. The learning rate was multiplied by 0.2 after $60, 120$, and 160 epochs respectively. We use WideResNet-28-10 in Chapter 4 and WideResNet-26-10 in Chapter 7.

3. **Densenet-100/121** The DenseNet-100 is a standard 100-layered densenet with Batchnorm and ReLU and has a total of $800,032$ trainable parameters.

We train the network on CIFAR10/100 for a total of 350 epochs with SGD using an initial learning rate of 0.1, which is multiplied by 0.1 after 150 and 250 epochs respectively, a weight decay of $1e-4$, and a momentum of 0.9. We use Densenet-100 in Chapter 4 and Densenet-121 in Chapters 5 and 7.

4. **VGG19** The VGG19 model is the standard 19-layered VGG model with Batchnorm and ReLU. It has three fully connected (FC) layers after sixteen convolution layers. It has a total of $20,548,392$ trainable parameters and is trained with SGD with a momentum of 0.9 and a weight decay of $5e-4$. The initial learning rate is 0.1 and is multiplied by 0.1 after 150 and 250 epochs respectively. We use this network for experiments in Chapters 4 and 5. For the shattering experiments in Chapter 4, we used the same architecture and the same training recipe except the initial learning rate, which was deceased to 0.01 as the model failed to learn the random labels with a large learning rate.

5. **AlexNet** The Alexnet model is the standard ALexNet model with $4,965,092$ trainable parameters. It was trained with SGD, with a momentum of 0.9, with an initial learning rate is 0.01, which is multiplied by 0.1 after 150 and 250 epochs respectively. The optimiser was further augmented with a weight decay rate of $5e-4$. We use this network for the experiments in Chapter 5.

6. **GAN Architecture** The model architecture for both the generator and the discriminator was chosen to be a 32 layered ResNet [110] (similar to ResNet-18 above) due to its previous superior performance in other works [182]. We use Adam optimiser [141] which depends on three main hyper-parameters α- the initial learning rate, β_1- the first order moment decay rate and β_2- the second order moment decay rate. We cross-validate these parameters in the set $\alpha \in \{0.0002, 0.0005\}$, $\beta_1 \in \{0, 0.5\}$, $\beta_2 \in \{0.9, 0.999\}$ and chose $\alpha = 0.0002$, $\beta_1 = 0.0$ and $\beta_2 = 0.999$ which performed consistently well in all of the experiments.

7. **Neural Divergence Setup** We train a new classifier inline with the architec-

ture in Gulrajani et al. [98]. It includes three convolution layers with 16, 32 and 64 channels, a kernel size of $5\times$ 5 and a stride of 2. Each of these layers are followed by a Swish activation [215] and then finally a linear layer that gives a single output. The network is initialised using normal distribution with zero mean and the standard deviation of 0.02, and trained using Adam optimiser with $\alpha = 0.0002,\ \beta_1 = 0.,\ \beta_2 = 0.9$ for a total of $100,000$ iterations with mini-batch of 128 generated samples and 128 samples from the test-set[5]. We use the standard WGAN-GP loss function, $\log\left(1+\exp\left(f\left(\mathbf{x}_{\text{fake}}\right)\right)\right)+\log\left(1+\exp\left(-\mathbf{x}_{\text{real}}\right)\right)$, where f represents the network described above. Finally, we generate 1 Million samples from the generator and report the average $\log\left(1+\exp\left(f\left(\mathbf{x}_{\text{fake}}\right)\right)\right)$ over these samples. Higher average value implies better generation as the network in this case is unable to distinguish the generated and the real samples.

[5]For CelebA, we used the training set.

C. Appendix for Chapter 4

C.1 Proofs for Section 4.2.2

Proof of Lemma 1. As mentioned in Equation (4.3), local lipschitzness of a function f at $\mathbf{x}$ is equal to the jacobian at that point i.e. $L_l(\mathbf{x}) = \|\mathrm{J_f}(\mathbf{x})\|_{p,q}^{\mathrm{op}}$

$$J_f(\mathbf{x}) = \frac{\partial f\left(\mathbf{x}\right)}{\partial \mathbf{x}} := \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}} \frac{\partial \phi_1(\mathbf{z}_1)}{\partial \mathbf{z}_1} \cdots \frac{\partial \mathbf{z}_L}{\partial \mathbf{a}_{L-1}} \frac{\partial \phi_L(\mathbf{z}_L)}{\partial \mathbf{z}_L}. \tag{C.1}$$

Using $\frac{\partial \mathbf{z}_i}{\partial \mathbf{a}_{i-1}} = \mathbf{W}_i$ where $\mathbf{z}_i$ is the pre-activation vector of the i^{th} layer, $\mathbf{W}_i$ represents the linear operator of the i^{th} layer, and $\mathbf{a}_i = \phi_i\left(\mathbf{z}_i\right)$ is the post-activation vector of the i^{th} layer. By sub-multiplicativity of the matrix norms:

$$\|\mathrm{J_f}(\mathbf{x})\|_{p,q}^{\mathrm{op}} \leq \|\mathbf{W}_1\|_{p,q}^{\mathrm{op}} \left\| \frac{\partial \phi_1(\mathbf{z}_1)}{\partial \mathbf{z}_1} \right\| \cdots \|\mathbf{W}_{\mathrm{L}}\|_{p,q}^{\mathrm{op}} \left\| \frac{\partial \phi_{\mathrm{L}}(\mathbf{z}_{\mathrm{l}})}{\partial \mathbf{z}_{\mathrm{L}}} \right\|. \tag{C.2}$$

Note, most commonly used activation functions $\phi(.)$ such as ReLU, sigmoid, tanh and maxout are known to have Lipschitz constant less than or equal to 1 Thus, the upper bound can further be written only using the operator norms of the intermediate matrices as

$$L_l(\mathbf{x}) \leq \|\mathrm{J_f}(\mathbf{x})\|_{p,q}^{\mathrm{op}} \leq \|\mathbf{W}_{\mathrm{L}}\|_{p,q}^{\mathrm{op}} \cdots \|\mathbf{W}_1\|_{p,q}^{\mathrm{op}} \tag{C.3}$$

As this upper bound is independent of the data point $\mathbf{x}$, this is also equal to the local lipschitz constant of f.

$$L_l = \max_{\mathbf{x}} L_l\left(\mathbf{x}\right) \leq \|\mathbf{W}_{\mathrm{L}}\|_{p,q}^{\mathrm{op}} \cdots \|\mathbf{W}_1\|_{p,q}^{\mathrm{op}}$$

□

Proof of Lemma 2. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function on an open set containing $\mathbf{x}_i$ and $\mathbf{x}_j$ such that $\mathbf{x}_i \neq \mathbf{x}_j$. By applying fundamental theorem of calculus

$$\begin{aligned}
\left|f\left(\mathbf{x}_i\right) - f\left(\mathbf{x}_j\right)\right| &= \left| \int_0^1 \nabla f\left(\mathbf{x}_i + \theta\left(\mathbf{x}_j - \mathbf{x}_i\right)\right)^\top \left(\mathbf{x}_j - \mathbf{x}_i\right) \partial\theta \right| \\
&\overset{(a)}{\leq} \int_0^1 \left| \nabla f\left(\mathbf{x}_i + \theta\left(\mathbf{x}_j - \mathbf{x}_i\right)\right)^\top \left(\mathbf{x}_j - \mathbf{x}_i\right) \right| \partial\theta \\
&\overset{(b)}{\leq} \int_0^1 \|\nabla \mathrm{f}\left(\mathbf{x}_{\mathrm{i}} + \theta\left(\mathbf{x}_{\mathrm{j}} - \mathbf{x}_{\mathrm{i}}\right)\right)\|_q \|\left(\mathbf{x}_{\mathrm{j}} - \mathbf{x}_{\mathrm{i}}\right)\|_p \partial\theta \\
&\leq \int_0^1 \max_{\theta \in (0,1)} \|\nabla \mathrm{f}\left(\mathbf{x}_{\mathrm{i}} + \theta\left(\mathbf{x}_{\mathrm{j}} - \mathbf{x}_{\mathrm{i}}\right)\right)\|_q \|\left(\mathbf{x}_{\mathrm{j}} - \mathbf{x}_{\mathrm{i}}\right)\|_p \partial\theta
\end{aligned}$$

$$= \max_{\theta\in(0,1)} \|\nabla \mathrm{f}\left(\mathbf{x}_\mathrm{i} + \theta\left(\mathbf{x}_\mathrm{j} - \mathbf{x}_\mathrm{i}\right)\right)\|_q \|(\mathbf{x}_\mathrm{j} - \mathbf{x}_\mathrm{i})\|_p \int_0^1 \partial\theta$$

$$\therefore \frac{|f\left(\mathbf{x}_i\right) - f\left(\mathbf{x}_j\right)|}{\|(\mathbf{x}_\mathrm{j} - \mathbf{x}_\mathrm{i})\|_p} \leq \max_{\theta\in(0,1)} \|\nabla \mathrm{f}\left(\mathbf{x}_\mathrm{i} + \theta\left(\mathbf{x}_\mathrm{j} - \mathbf{x}_\mathrm{i}\right)\right)\|_q = \max_{\mathbf{x}\in Conv\left(\mathbf{x}_i, \mathbf{x}_j\right)} \|\nabla \mathrm{f}\left(\mathbf{x}\right)\|_q.$$

The inequality (a) is due to Jensen's Inequality (Inequality 3) and inequality (b) is due to Hölder's inequality (Inequality 4). □

C.2 Proofs for Section 4.3.2

Proof for Optimal Stable Rank Normalization. (Main Theorem)

Proof. Here we provide the proof of Theorem 1 for all the three cases with optimality and uniqueness guarantees. Let $\widehat{\mathbf{W}}_k$ be the optimal solution to the problem for any of the two cases. From Lemma 5, the SVD of $\mathbf{W}$ and $\widehat{\mathbf{W}}_k$ can be written as $\mathbf{W} = \mathbf{U}\Sigma\mathbf{V}^\top$ and $\widehat{\mathbf{W}}_k = \mathbf{U}\Lambda\mathbf{V}^\top$, respectively. Then, $L = \left\|\mathbf{W} - \widehat{\mathbf{W}}_\mathrm{k}\right\|_\mathrm{F}^2 = \langle \Sigma - \Lambda, \Sigma - \Lambda\rangle_\mathrm{F}$. From now onwards, we denote Σ and Λ as vectors consisting of the diagonal entries, and $\langle ., .\rangle$ as the vector inner product [1].

Proof for Case (a): In this case, there is no constraint enforced to preserve any of the singular values of the given matrix while obtaining the new one. The only constraint is that the new matrix should have the stable rank of r. Let us assume $\Sigma = (\sigma_1, \cdots, \sigma_p)$, $\Sigma_2 = (\sigma_2, \cdots, \sigma_p)$, $\Lambda = (\lambda_1, \cdots, \lambda_p)$ and $\Lambda_2 = (\lambda_2, \cdots, \lambda_p)$. Using these notations, we can write L as:

$$\begin{aligned} L &= \langle \Sigma, \Sigma\rangle + \langle \Lambda, \Lambda\rangle - 2\langle \Sigma, \Lambda\rangle \\ &= \langle \Sigma, \Sigma\rangle + \lambda_1^2 + \langle \Lambda_2, \Lambda_2\rangle - 2\sigma_1\lambda_1 - 2\langle \Sigma_2, \Lambda_2\rangle \end{aligned} \tag{C.4}$$

Using the stable rank constraint $\mathrm{srank}(\widehat{\mathbf{W}}_k) = r$, which is $r = 1 + \dfrac{\sum_{j=2}^p \lambda_j^2}{\lambda_1^2}$.

Case for $\mathbf{r > 1}$ If $r > 1$ we obtain the following equality constraint, making the problem non-convex.

$$\lambda_1^2 = \frac{\langle \Lambda_2, \Lambda_2\rangle}{r-1} \tag{C.5}$$

However, we will show that the solution we obtain is optimal and unique. Substituting Equation (C.5) into Equation (C.4) we get

$$L = \langle \Sigma, \Sigma\rangle + \frac{\langle \Lambda_2, \Lambda_2\rangle}{r-1} + \langle \Lambda_2, \Lambda_2\rangle - 2\sigma_1\sqrt{\frac{\langle \Lambda_2, \Lambda_2\rangle}{r-1}} - 2\langle \Sigma_2, \Lambda_2\rangle \tag{C.6}$$

[1] $\langle ., .\rangle_\mathrm{F}$ represents the Frobenius inner product of two matrices, which in the case of diagonal matrices is the same as the inner product of the diagonal vectors.

Setting $\dfrac{\partial L}{\partial \Lambda_2} = 0$ to get the family of critical points

$$\frac{2\Lambda_2}{r-1} + 2\Lambda_2 - \frac{4\sigma_1\Lambda_2}{2\sqrt{(r-1)\langle\Lambda_2,\Lambda_2\rangle}} - 2\Sigma_2 = 0$$

$$\implies \Sigma_2 = \Lambda_2\left(\frac{1}{r-1} + 1 - \frac{\sigma_1}{1\sqrt{(r-1)\langle\Lambda_2,\Lambda_2\rangle}}\right) \tag{C.7}$$

$$\implies \frac{\Sigma_2[i]}{\lambda_2[i]} = \left(\frac{1}{r-1} + 1 - \frac{\sigma_1}{1\sqrt{(r-1)\langle\Lambda_2,\Lambda_2\rangle}}\right) = \frac{1}{\gamma_2} \quad \forall\ 1 \le i \le p \tag{C.8}$$

As the R.H.S. of C.8 is independent of i, the above equality implies that all the critical points of Equation (C.6) are a scalar multiple of Σ_2, implying, $\Lambda_2 = \gamma_2\Sigma_2$. Note that the domain of Λ_2 are all strictly positive vectors and thus, we can ignore the critical point at $\Lambda_2 = \mathbf{0}$. Substituting this into Equation (C.7) we obtain

$$\Sigma_2 = \gamma_2\Sigma_2\left(\frac{1}{r-1} + 1 - \frac{\sigma_1}{\gamma_2\sqrt{(r-1)\langle\Sigma_2,\Sigma_2\rangle}}\right)$$

Using the fact that $\langle\Sigma_2,\Sigma_2\rangle = \|\mathbf{S}_2\|_{\mathrm{F}}^2$ in the above equality and with some algebraic manipulations, we obtain $\gamma_2 = \frac{\gamma+r-1}{r}$ where, $\gamma = \frac{\sqrt{r-1}\sigma_1}{\|\mathbf{S}_2\|_{\mathrm{F}}}$. Note, $r \ge 1$, $\gamma \ge 0$, and $\Sigma \ge 0$, implying, $\Lambda_2 = \gamma_2\Sigma_2 \ge 0$.

Local minima: Now, we will show that Λ_2 is indeed a minima of Equation (C.6). To show this, we compute the hessian of L. Recall that

$$\frac{\partial L}{\partial \Lambda} = \frac{2r}{r-1}\Lambda - \frac{2\sigma_1\Lambda}{\sqrt{(r-1)\|\Lambda\|_2^2}} - 2\Sigma_2$$

$$\mathbf{H} = \frac{\partial^2 L}{\partial^2 \Lambda} = \frac{2r}{r-1}\mathbf{I} - \frac{2\sigma_1}{\sqrt{(r-1)}\|\Lambda\|_2^2}\left(\|\Lambda\|_2\mathbf{I} - \frac{1}{\|\Lambda\|_2}\Lambda\Lambda^\top\right)$$

$$= 2\left(\frac{r}{r-1} - \frac{\sigma_1\|\Lambda\|_2}{\sqrt{(r-1)}\|\Lambda\|_2^2}\right)\mathbf{I} + \frac{2\sigma_1}{\sqrt{r-1}\|\Lambda\|_2^3}\left(\Lambda\Lambda^\top\right)$$

Now we need to show that $\mathbf{H}$ at the solution Λ_2 is PSD i.e. $\forall\,\mathbf{x} \in \mathbb{R}^{p-1}$, $\mathbf{x}^\top\mathbf{H}\left(\boldsymbol{\Lambda_2}\right)\mathbf{x} \ge 0$

$$\mathbf{x}^\top\mathbf{H}\mathbf{x} = 2\left(\frac{r}{r-1} - \frac{\sigma_1\|\Lambda\|_2}{\sqrt{(r-1)}\|\Lambda\|_2^2}\right)\left\|\mathbf{x}_2^2\right\| + \frac{2\sigma_1}{\sqrt{r-1}\|\Lambda\|_2^3}\mathbf{x}^\top\left(\Lambda\Lambda^\top\right)\mathbf{x}$$

$$\overset{(a)}{\ge} 2\left(\frac{r}{r-1} - \frac{\sigma_1\|\Lambda\|_2}{\sqrt{(r-1)}\|\Lambda\|_2^2}\right)\|\mathbf{x}\|_2^2$$

$$\overset{(b)}{\ge} 2\left(\frac{r}{r-1} - \frac{\sigma_1}{(r-1)\lambda_1}\right)\|\mathbf{x}\|_2^2 \overset{(c)}{=} \frac{2r}{r-1}\left(1 - \frac{\gamma}{(\gamma+r-1)}\right)\|\mathbf{x}\|_2^2$$

$$\overset{(d)}{\geq} \frac{2r}{r-1}\left(1-\frac{1}{(1+r-1)}\right)\|\mathbf{x}\|_2^2 = 2\|\mathbf{x}\|_2^2 \geq 0$$

Here (a) is due to the fact that the matrix $\Lambda\Lambda^\top$ is an outer product matrix and is hence PSD (see Equation (A.5)). (b) follows due to Equation (C.5) and (c) follows by substituting $\lambda_1 = \gamma_1\sigma_1$ and then the value of γ_1. Finally (d) follows as $\left(1-\frac{\gamma}{(\gamma+r-1)}\right)$ is decreasing with respect to γ and we know that $\gamma < 1$ due to the assumption that srank$(\mathbf{W}) < r$. Thus, we can substitute $\gamma = 1$ to find the minimum value of the expression. This concludes our proof that Λ_2 is indeed a local minima of L.

Uniqueness: The uniqueness of Λ_2 as a solution to Equation (C.6) is shown in Proposition 8 and is also guaranteed by the fact that γ_2 has a unique value. Using $\Lambda_2 = \gamma_2\Sigma_2$ and $\lambda_1 = \gamma_1\sigma_1$ in Equation (C.5), we obtain a unique solution $\gamma_1 = \frac{\gamma_2}{\gamma}$.

Now, we need to show that it is also an unique solution to Theorem 1.

For all solutions to Theorem 1 that have singular vectors which are different than that of $\mathbf{W}$, by Lemma 5, the matrix formed by replacing the singular vectors of the solution with that of $\mathbf{W}$ is also a solution. Thus, if there were a solution with different singular values than $\widehat{\mathbf{W}}_k$, it should have appeared as a solution to Equation (C.6). However, we have shown that Equation (C.6) has a unique solution.

Now, we need to show that among all matrices with the same singular values as that of $\widehat{\mathbf{W}}_k$, $\widehat{\mathbf{W}}_k$ is strictly better in terms of $\left\|\mathbf{W}-\widehat{\mathbf{W}}_{\mathrm{k}}\right\|$. This requires a further assumption that every non-zero singular value of Λ_2 has a multiplicity of 1 i.e. they are all distinctly unique. Intuitively, this doesn't allow to create a different matrix by simply interchanging the singular vectors associated with the equal singular values. As the elements of Σ_2 are distinct, the elements of $\Lambda_2 = \gamma_2\Sigma_2$ are also distinct and thus by the second part of Lemma 5, $\widehat{\mathbf{W}}_k$ is strictly better, in terms of $\left\|\mathbf{W}-\widehat{\mathbf{W}}_{\mathrm{k}}\right\|$, than all matrices which have the same singular values as that of $\widehat{\mathbf{W}_k}$. This concludes our discussion on the uniqueness of the solution.

Case for r = 1: Substituting $r = 1$ in the constraint $r = 1 + \frac{\sum_{j=2}^p \lambda_j^2}{\lambda_1^2}$ we get

$$r - 1 = \frac{\sum_{j=2}^p \lambda_j^2}{\lambda_1^2} = 0 \implies \sum_{j=2}^p \lambda_j^2 = 0$$

As it is a sum of squares, each of the individual elements is also zero i.e. $\lambda_j = 0\ \forall 2 \leq j \leq p$. Substituting this into Equation (C.4), we get the following quadratic equation in λ_1

$$L = \langle \Sigma, \Sigma \rangle + \lambda_1^2 - 2\sigma_1\lambda_1 \tag{C.9}$$

which is minimised at $\lambda_1 = \sigma_1$, thus proving that $\gamma_1 = 1$ and $\gamma_2 = 0$.

Proof for Case (b): In this case, the constraints are meant to preserve the top k singular values of the given matrix while obtaining the new one. Let $\Sigma_1 = (\sigma_1, \cdots, \sigma_k)$, $\Sigma_2 = (\sigma_{k+1}, \cdots, \sigma_p)$, $\Lambda_1 = (\lambda_1, \cdots, \lambda_k)$, $\Lambda_2 = (\lambda_{k+1}, \cdots, \lambda_p)$. Since satisfying all the constraints imply $\Sigma_1 = \Lambda_1$, thus, $L := \left\| \mathbf{W} - \widehat{\mathbf{W}}_{\mathrm{k}} \right\|_{\mathrm{F}}^2 = \langle \Sigma_2 - \Lambda_2, \Sigma_2 - \Lambda_2 \rangle$. From the stable rank constraint $\mathrm{srank}(\widehat{\mathbf{W}}_k) = r$, we have

$$\begin{aligned} r &= \frac{\langle \Lambda_1, \Lambda_1 \rangle + \langle \Lambda_2, \Lambda_2 \rangle}{\lambda_1^2} \\ \therefore \quad \langle \Lambda_2, \Lambda_2 \rangle &= r\lambda_1^2 - \langle \Lambda_1, \Lambda_1 \rangle = r\sigma_1^2 - \langle \Sigma_1, \Sigma_1 \rangle \end{aligned} \tag{C.10}$$

The above equality constraint makes the problem non-convex. Thus, we relax it to $\mathrm{srank}(\widehat{\mathbf{W}}_k) \leq r$ to make it a convex problem and show that the optimality is achieved with equality. Let $r\sigma_1^2 - \langle \Sigma_1, \Sigma_1 \rangle = \eta$. Then, the relaxed problem can be written as

$$\begin{aligned} &\min_{\Lambda_2 \in \mathbb{R}^{p-k}} L := \langle \Sigma_2 - \Lambda_2, \Sigma_2 - \Lambda_2 \rangle \\ &\text{s.t.} \quad \Lambda_2 \geq 0, \langle \Lambda_2, \Lambda_2 \rangle \leq \eta. \end{aligned}$$

We introduce the Lagrangian dual variables $\Gamma \in \mathbb{R}^{p-k}$ and μ corresponding to the positivity and the stable rank constraints, respectively. The Lagrangian can then be written as

$$\mathcal{L}\left(\Lambda_2, \Gamma, \mu\right)_{\Gamma \geq \mathbf{0}, \mu \geq 0} = \langle \Sigma_2 - \Lambda_2, \Sigma_2 - \Lambda_2 \rangle + \mu \left(\langle \Lambda_2, \Lambda_2 \rangle - \eta \right) - \langle \Gamma, \Lambda_2 \rangle \tag{C.11}$$

Using the primal optimality condition $\dfrac{\partial \mathcal{L}}{\partial \Lambda_2} = \mathbf{0}$, we obtain

$$\begin{aligned} &2\Lambda_2 - 2\Sigma_2 + 2\mu\Lambda_2 - \Gamma = \mathbf{0} \\ \implies \quad &\Lambda_2 = \frac{\Gamma + 2\Sigma_2}{2\left(1 + \mu\right)} \end{aligned} \tag{C.12}$$

Using the above condition on Λ_2 with the constraint $\langle \Lambda_2, \Lambda_2 \rangle \leq \eta$, combined with the stable rank constraint of the given matrix $\mathbf{W}$ that comes with the problem definition, $\mathrm{srank}(\mathbf{W}) > r$ (which implies $\langle \Sigma_2, \Sigma_2 \rangle > \eta$), the following inequality must be satisfied for any $\Gamma \geq 0$

$$1 < \frac{\langle \Sigma_2, \Sigma_2 \rangle}{\eta} \leq \frac{\langle \Gamma + \Sigma_2, \Gamma + \Sigma_2 \rangle}{\eta} \leq (1 + \mu)^2 \tag{C.13}$$

For the above inequality to satisfy, the dual variable μ must be greater than zero, implying, $\langle \Lambda_2, \Lambda_2 \rangle - \eta$ must be zero for the complementary slackness to satisfy. Using

this with the optimality condition Equation (C.12) we obtain

$$(1+\mu)^2 = \frac{\langle \Gamma + 2\Sigma_2, \Gamma + 2\Sigma_2 \rangle}{4\eta}$$

Substituting the above solution back into the primal optimality condition we get

$$\Lambda_2 = (\Gamma + 2\Sigma_2) \frac{\sqrt{\eta}}{\sqrt{\langle \Gamma + 2\Sigma_2, \Gamma + 2\Sigma_2 \rangle}} \tag{C.14}$$

Finally, we use the complimentary slackness condition $\Gamma \odot \Lambda_2 = \mathbf{0}$[2] to get rid of the dual variable Γ as follows

$$\Gamma \odot (\Gamma + 2\Sigma_2) \frac{\sqrt{\eta}}{\sqrt{\langle \Gamma + 2\Sigma_2, \Gamma + 2\Sigma_2 \rangle}} = \mathbf{0}$$

It is easy to see that the above condition is satisfied only when $\Gamma = \mathbf{0}$ as $\Sigma_2 \geq \mathbf{0}$ and $\eta > 0$. Therefore, using $\Gamma = \mathbf{0}$ in Equation (C.14) we obtain the optimal solution of Λ_2 as

$$\Lambda_2 = \frac{\sqrt{\eta}}{\sqrt{\langle \Sigma_2, \Sigma_2 \rangle}} \Sigma_2 = \frac{\sqrt{r\sigma_1^2 - \|\mathbf{S}_1\|_{\mathrm{F}}^2}}{\|\mathbf{S}_2\|_{\mathrm{F}}^2} \Sigma_2 = \gamma \Sigma_2 \tag{C.15}$$

Proof for Case (c): The monotonicity of $\left\| \widehat{\mathbf{W}}_{\mathrm{k}} - \mathbf{W} \right\|_{\mathrm{F}}$ for $k \geq 1$ is shown in Lemma 4. □

Note that by the assumption that $\mathrm{srank}(\mathbf{W}) < r$, we can say that $\gamma < 1$. Therefore in all the cases $\gamma_2 < 1$. Let us look at the required conditions for $\gamma_1 \geq 1$ to hold. When $k \geq 1$, $\gamma_1 = 1$ holds. When $k = 0$, for $\gamma_1 > 1$ to be true, $\gamma_2 < \gamma$ should hold, implying, $(\gamma - 1) < r\,(\gamma - 1)$, which is always true as $r > 1$ (by the definition of stable rank).

Lemma 4 (Monotoncity of Solutions w.r.t. the partitioning index)**.** *For $k \geq 1$, the solution to the optimisation problem Equation* (4.7) *obtained using Theorem 1 is closest to the original matrix* $\mathbf{W}$ *in terms of Frobenius norm when only the spectral norm is preserved, implying, $k = 1$.*

Proof. For a given matrix $\mathbf{W}$ and a partitioning index $k \in \{1, \cdots, p\}$, let $\widehat{\mathbf{W}}_k = \mathbf{S}_1^k + \gamma \mathbf{S}_2^k$ be the matrix obtained using Theorem 1. We use the superscript k along with $\mathbf{S}_1$ and $\mathbf{S}_2$ to denote that this refers to the particular solution of $\widehat{\mathbf{W}}_k$. Plugging the value of γ and using the fact that $\left\|\mathbf{S}_2^{\mathrm{k}}\right\|_{\mathrm{F}} \neq 0$, we can write

$$\begin{aligned} \left\| \mathbf{W} - \widehat{\mathbf{W}}_{\mathrm{k}} \right\|_{\mathrm{F}} &= (1-\gamma) \left\|\mathbf{S}_2^{\mathrm{k}}\right\|_{\mathrm{F}} \\ &= \left\|\mathbf{S}_2^{\mathrm{k}}\right\|_{\mathrm{F}} - \sqrt{r\sigma_1^2 - \left\|\mathbf{S}_1^{\mathrm{k}}\right\|_{\mathrm{F}}^2} \end{aligned}$$

[2] $\odot$ is the hadamard product

$$= \left\|\mathbf{S}_2^{\mathrm{k}}\right\|_{\mathrm{F}} - \sqrt{r\sigma_1^2 - \|\mathbf{W}\|_{\mathrm{F}}^2 + \left\|\mathbf{S}_2^{\mathrm{k}}\right\|_{\mathrm{F}}^2}.$$

Thus, $\left\|\mathbf{W} - \widehat{\mathbf{W}}_{\mathrm{k}}\right\|_{\mathrm{F}}$ can be written in a simplified form as $f(x) = x - \sqrt{a + x^2}$, where $x = \left\|\mathbf{S}_2^{\mathrm{k}}\right\|_{\mathrm{F}}$ and $a = r\sigma_1^2 - \|\mathbf{W}\|_{\mathrm{F}}^2$. Note, $a \leq 0$ as $1 \leq r \leq \mathrm{srank}(\mathbf{W})$, and $a + x^2 \geq 0$ because of the condition in Theorem 1. Under these settings, it is trivial to verify that f is a monotonically decreasing function of x. Using the fact that as the partition index k increases, x decreases, it is straightforward to conclude that the minimum of $f(x)$ is obtained at $k = 1$. □

Auxiliary Lemmas

Lemma A (Reproduced from Theorem 5 in Mirsky [178])**.** *For any two matrices* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ *with singular values as* $\sigma_1 \geq \cdots \geq \sigma_n$ *and* $\rho_1 \geq \cdots \geq \rho_n$*, respectively*

$$\|\mathbf{A} - \mathbf{B}\|_{\mathrm{F}}^2 \geq \sum_{i=1}^{n} (\sigma_i - \rho_i)^2$$

Proof. Consider the following symmetric matrices

$$\mathbf{X} = \begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{A}^\top & \mathbf{0} \end{bmatrix}, \mathbf{Y} = \begin{bmatrix} \mathbf{0} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{0} \end{bmatrix}, \mathbf{Z} = \begin{bmatrix} \mathbf{0} & \mathbf{A} - \mathbf{B} \\ (\mathbf{A} - \mathbf{B})^\top & \mathbf{0} \end{bmatrix}$$

Let $\tau_1 \geq \cdots \geq \tau_n$ be the singular values of $\mathbf{Z}$. Then the set of characteristic roots of $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$ in descending order are $\{\rho_1, \cdots, \rho_n, -\rho_n, \cdots, -\rho_1\}$, $\{\sigma_1, \cdots, \sigma_n, -\sigma_n, \cdots, -\sigma_1\}$, and $\{\tau_1, \cdots, \tau_n, -\tau_n, \cdots, -\tau_1\}$, respectively. By Lemma 2 in Wielandt [272]

$$[\sigma_1 - \rho_1, \cdots, \sigma_n - \rho_n, \rho_n - \sigma_n, \cdots, \rho_1 - \sigma_1] \preceq [\tau_1, \cdots \tau_n, -\tau_n, -\tau_1],$$

which implies that

$$\sum_{i=1}^{n} (\sigma_i - \rho_i)^2 \leq \sum_{i=1}^{n} \tau_i^2 = \|\mathbf{A} - \mathbf{B}\|_{\mathrm{F}}^2 \tag{C.16}$$

□

Lemma 5 (Minimisation wrt Frobenius Norm requires singular values only)**.** *Let* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ *where* $\mathrm{SVD}(\mathbf{A}) = \mathbf{U}\Sigma\mathbf{V}^\top$ *and* $\mathbf{B}$ *is the solution to the following problem*

$$\mathbf{B} = \underset{\mathrm{srank}(\mathbf{W}) = r}{\mathbf{argmin}} \|\mathbf{W} - \mathbf{A}\|_{\mathrm{F}}^2. \tag{C.17}$$

Then, $\mathrm{SVD}(\mathbf{B}) = \mathbf{U}\Lambda\mathbf{V}^\top$ *where* Λ *is a diagonal matrix with non-negative entries. Implying,* $\mathbf{A}$ *and* $\mathbf{B}$ *will have the same singular vectors.*

Proof. Let us assume that $\mathbf{Z} = \mathbf{S}\Lambda\mathbf{T}^\top$ is a solution to the problem C.17 where $\mathbf{S} \neq \mathbf{U}$ and $\mathbf{T} \neq \mathbf{V}$. Trivially, $\mathbf{X} = \mathbf{U}\Lambda\mathbf{V}^\top$ also lies in the feasible set as it satisfies

srank($\mathbf{X}$) = r (note stable rank only depends on the singular values). Using the fact that the Frobenius norm is invariant to unitary transformations, we can write $\|\mathbf{A} - \mathbf{X}\|_{\mathrm{F}}^2 = \left\|\mathbf{U}\left(\Sigma - \Lambda\right)\mathbf{V}^\top\right\|_{\mathrm{F}}^2 = \|(\Sigma - \Lambda)\|_{\mathrm{F}}^2$. Combining this with Lemma A, we obtain $\|\mathbf{A} - \mathbf{X}\|_{\mathrm{F}}^2 = \|(\Sigma - \Lambda)\|_{\mathrm{F}}^2 \leq \|\mathbf{A} - \mathbf{Z}\|_{\mathrm{F}}^2$. Since, $\mathbf{S} \neq \mathbf{U}$ and $\mathbf{T} \neq \mathbf{V}$, we can further change $\leq$ to a strict inequality $<$. This completes the proof.

Generally speaking, the optimal solution to problem C.17 with constraints depending only on the singular values (*e.g.* stable rank in this case) will have the same singular vectors as that of the original matrix. Further the inequality in Equation (C.16) can be converted into a strict inequality if neither of $\mathbf{A}$ and $\mathbf{B}$ have repeated singular values. □

Proposition 8. *Let* $\mathbf{y}_1 = a\mathbf{x}_1 + b\hat{\mathbf{x}}_1$ *and* $\mathbf{y}_2 = a\mathbf{x}_2 + b\hat{\mathbf{x}}_2$*, where* $\hat{\mathbf{x}}_1$ *and* $\hat{\mathbf{x}}_2$ *denotes the unit vectors. Then,* $\mathbf{y}_1 = \mathbf{y}_2$ *if* $\mathbf{x}_1 = \mathbf{x}_2$.

D. Appendix for Chapter 5

D.1 Proofs for Section 5.3.1

Proof of Theorem 3. We define a family of distribution $\mathcal{D}$, such that each distribution in $\mathcal{D}$ is supported on balls of radius r around (i, i) and $(i+1, i)$ for positive integers i. Either all the balls around (i, i) have the labels 1 and the balls around $(i+1, i)$ have the label 0 or vice versa. Figure 5.6(b) shows an example where the colours indicate the label.

Formally, for $r > 0$, $k \in \mathbb{Z}_+$, the (r, k)-1 bit parity class conditional model is defined over $(x, y) \in \mathbb{R}^2 \times \{0, 1\}$ as follows. First, a label y is sampled uniformly from $\{0, 1\}$, then and integer i is sampled uniformly from the set $\{1, \cdots, k\}$ and finally $\mathbf{x}$ is generated by sampling uniformly from the ℓ_2 ball of radius r around $(i + y, i)$.

In Lemma 6 we first show that a set of m points sampled iid from any distribution as defined above for $r < \frac{1}{2\sqrt{2}}$ is with probability 1 linear separable for any m. In addition, standard VC bounds show that any linear classifier that separates S_m for large enough m will have small test error. Lemma 6 also proves that there exists a range of γ, r such that for any distribution defined with r in that range, though it is possible to obtain a linear classifier with 0 training and test error, the minimum adversarial risk will be bounded away from 0.

However, while it is possible to obtain a linear classifier with 0 test error, all such linear classifiers has a large adversarial vulnerability. In Lemma 7, we show that there exists a different representation for this problem, which also achieves zero training and test error, and in addition has zero adversarial risk for a range of r and γ where the linear classifier's adversarial error was at least a non-zero positive constant.

□

Lemma 6 (Robustness of Linear Classifier)**.** *There exists universal constants γ_0, ρ, such that for any perturbation $\gamma > \gamma_0$, radius $r \geq \rho$, and $k \in \mathbb{Z}_+$, the following holds. Let $\mathcal{D}$ be the family of (r, k)- 1-bit parity class conditional model, $\mathcal{P} \in \mathcal{D}$ and $\mathcal{S}_n = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_1)\}$ be a set of n points sampled i.i.d. from $\mathcal{P}$.*

1) For any $n > 0$, S_n is linearly separable with probability 1 i.e. there exists a $h : (\mathbf{w}, w_0)$, $\mathbf{w} \in \mathbb{R}^2, w_0 \in \mathbb{R}$ such that the linear hyperplane $\mathbf{x} \to \mathbf{w}^\top \mathbf{x} + w_0$ separates $\mathcal{S}_n$ with probability 1:

$$\forall\, (\mathbf{x}, y) \in \mathcal{S}_n \quad z\left(\mathbf{w}^\top \mathbf{x} + w_0\right) > 0 \quad \text{where } z = 2y - 1$$

2) Further there exists an universal constant c such that for any $\epsilon, \delta > 0$ with

probability $1-\delta$ *for any* $\mathcal{S}_n$ *with* $n = c\frac{1}{\epsilon^2}\log\frac{1}{\delta}$*, any linear classifier* $\tilde{h}$ *that separates* $\mathcal{S}_n$ *has* $\mathcal{R}(\tilde{h};\mathcal{P}) \leq \epsilon$*.*

3) Let $h : (\mathbf{w}, w_0)$ *be any linear classifier that has* $\mathcal{R}(h;\mathcal{P}_P) = 0$*. Then,* $\mathcal{R}_{\text{Adv},\gamma}(h;\mathcal{P}) > 0.0005$*.*

We will prove the first part for any $r < \frac{1}{2\sqrt{2}}$ by constructing a $\mathbf{w}, w_0$ such that it satisfies the constraints of linear separability. Let $\mathbf{w} = (1,-1)$, $w_0 = -0.5$. Consider any point $(\mathbf{x}, y) \in \mathcal{S}_n$ and $z = 2y - 1$. Converting to the polar coordinate system there exists a $\theta \in [0, 2\pi], j \in [0, \cdots, k]$ such that $\mathbf{x} = \left(j + \frac{z+1}{2} + r\cos(\theta), j + r\sin(\theta)\right)$

$$
\begin{aligned}
z\left(\mathbf{w}^\top\mathbf{x} + w_0\right) &= z\left(j + \frac{z+1}{2} + r\cos(\theta) - j - r\sin(\theta) - 0.5\right) && \mathbf{w} = (1,-1)^\top \\
&= z\left(\frac{z}{2} + 0.5 + r\cos(\theta) - r\sin(\theta) - 0.5\right) \\
&= \frac{1}{2} + zr\left(\cos(\theta) - \sin(\theta)\right) && |\cos(\theta) - \sin(\theta)| < \sqrt{2} \\
&> \frac{1}{2} - r\sqrt{2} \\
&> 0 && r < \frac{1}{2\sqrt{2}}
\end{aligned}
$$

Part 2 follows from a simple application of VC bounds for linear classifiers.

Let the universal constants γ_0, ρ be 0.02 and $\frac{1}{2\sqrt{2}} - 0.008$ respectively. Note that there is nothing special about this constants except that *some* constant is required to bound the adversarial risk away from 0. Now, consider a distribution $\mathcal{P}$ 1-bit parity model where the radius of each ball is at least ρ. This is smaller than $\frac{1}{2\sqrt{2}}$ and thus satisfies the linear separability criterion.

Consider h to be a hyper-plane that has 0 test error. Let the ℓ_2 radius of adversarial perturbation be $\gamma > \gamma_0$. The region of each circle that will be vulnerable to the attack will be a circular segment with the chord of the segment parallel to the hyper-plane. Let the minimum height of all such circular segments be r_0. Thus, $\mathcal{R}_{\text{Adv},\gamma}(h;\mathcal{P})$ is greater than the mass of the circular segment of radius r_0. Let the radius of each ball in the support of $\mathcal{P}$ be r.

Using the fact that h has zero test error; and thus classifies the balls in the support of $\mathcal{P}$ correctly and simple geometry

$$
\begin{aligned}
\frac{1}{\sqrt{2}} &\geq r + (\gamma - r_0) + r \\
r_0 &\geq 2r + \gamma - \frac{1}{\sqrt{2}}
\end{aligned} \tag{D.1}
$$

To compute $\mathcal{R}_{\text{Adv},\gamma}(h;\mathcal{P})$ we need to compute the ratio of the area of a circular segment of height r_0 of a circle of radius r to the area of the circle. The ratio can

be written

$$A\left(\frac{r_0}{r}\right) = \frac{cos^{-1}\left(1-\frac{r_0}{r}\right) - \left(1-\frac{r_0}{r}\right)\sqrt{2\frac{r_0}{r} - \frac{r_0^2}{r^2}}}{\pi} \tag{D.2}$$

As Equation (D.2) is increasing with $\frac{r_0}{r}$, we can evaluate

$$\begin{aligned}
\frac{r_0}{r} &\geq \frac{2r - \frac{1}{\sqrt{2}} + \gamma}{r} && \text{Using } \textit{Equation} \text{ (D.1)} \\
&\geq 2 - \frac{\frac{1}{\sqrt{2}} - 0.02}{r} && \gamma > \gamma_0 = 0.02 \\
&\geq 2 - \frac{\frac{1}{\sqrt{2}} - 0.02}{\frac{1}{\sqrt{2}} - 0.008} > 0.01 && r > \rho = \frac{1}{2\sqrt{2}} - 0.008
\end{aligned}$$

Substituting $\frac{r_0}{r} > 0.01$ into Eq. Equation (D.2), we get that $A\left(\frac{r_0}{r}\right) > 0.0005$. Thus, for all $\gamma > 0.02$, we have $\mathcal{R}_{\text{Adv},\gamma}(h;\mathcal{P}) > 0.0005$.

Lemma 7 (Robustness of parity classifier)**.** *There exists a concept class $\mathcal{H}$ such that for any $\gamma \in \left[\gamma_0, \gamma_0 + \frac{1}{8}\right]$, $k \in \mathbb{Z}_+$, $\mathcal{P}$ being the corresponding (ρ, k) 1-bit parity class distribution where ρ, γ_0 are the same as in Lemma 6 there exists $g \in \mathcal{H}$ such that*

$$\mathcal{R}(g;\mathcal{P}) = 0 \qquad \mathcal{R}_{\text{Adv},\gamma}(g;\mathcal{P}) = 0$$

Proof of Lemma 7. We will again provide a proof by construction. Consider the following class of concepts $\mathcal{H}$ such that $g_b \in \mathcal{H}$ is defined as

$$g\left((x_1, x_2)^\top\right) = \begin{cases} 1 & \text{if } [x_1] + [x_2] = b \,(\text{mod } 2) \\ 1 - b & \text{o.w.} \end{cases} \tag{D.3}$$

where $[x]$ rounds x to the nearest integer and $b \in \{0, 1\}$. In Figure 5.6(b), the green staircase-like classifier belongs to this class. Consider the classifier g_1. Note that by construction $\mathcal{R}(g_1;\mathcal{P}) = 0$. The decision boundary of g_1 that are closest to a ball in the support of $\mathcal{P}$ centred at (a, b) are the lines $x = a \pm 0.5$ and $y = b \pm 0.5$.

As $\gamma < \gamma_0 + \frac{1}{8}$, the adversarial perturbation is upper bounded by $\frac{1}{50} + \frac{1}{8}$. The radius of the ball is upper bounded by $\frac{1}{2\sqrt{2}}$, and as we noted the center of the ball is at a distance of 0.5 from the decision boundary. If the sum of the maximum adversarial perturbation and the maximum radius of the ball is less than the minimum distance of the center of the ball from the decision boundary, then the adversarial error is 0. Substituting the values,

$$\frac{1}{50} + \frac{1}{8} + \frac{1}{2\sqrt{2}} < 0.499 < \frac{1}{2}$$

This completes the proof. □

D.2 Proofs for Section 5.3.2

Proof of Theorem 4. We will provide a constructive proof to this theorem by constructing a distribution $\mathcal{D}$, two concept classes $\mathcal{C}$ and $\mathcal{H}$ and provide the ERM algorithms to learn the concepts and then use Lemmas 8 and 9 to complete the proof.

Distribution: Consider the family of distribution $\mathcal{D}^n$ such that $\mathcal{D}_{S,\zeta} \in \mathcal{D}^n$ is defined on $\mathcal{X}_\zeta \times \{0,1\}$ for $S \subseteq \{1,\cdots,n\}, \zeta \subseteq \{1,\cdots,2^n-1\}$ such that the support of $\mathcal{X}_\zeta$ is a union of intervals.

$$\operatorname{supp}(\mathcal{X})_\zeta = \bigcup_{j\in\zeta} I_j \text{ where } I_j := \left(j - \frac{1}{4}, j + \frac{1}{4}\right) \tag{D.4}$$

We consider distributions with a relatively small support i.e. where $|\zeta| = \mathcal{O}(n)$. Each sample $(\mathbf{x}, y) \sim \mathcal{D}_{S,\zeta}$ is created by sampling $\mathbf{x}$ uniformly from $\mathcal{X}_\zeta$ and assigning $y = c_S(\mathbf{x})$ where $c_S \in \mathcal{C}$ is defined below (Equation (D.5)). We define the family of distributions $\mathcal{D} = \bigcup_{n\in\mathbb{Z}_+} \mathcal{D}^n$. Finally, we create $\mathcal{D}^\eta_{S,\zeta}$ -a noisy version of $\mathcal{D}_{S,\zeta}$, by flipping y in each sample (x, y) with probability $\eta < \frac{1}{2}$. Samples from $\mathcal{D}_{S,\zeta}$ can be obtained using the example oracle $\mathrm{EX}(\mathcal{D}_{S,\zeta})$ and samples from the noisy distribution can be obtained through the noisy oracle $\mathrm{EX}^\eta(\mathcal{D}_{S,\zeta})$

Concept Class $\mathcal{C}$: We define the concept class $\mathcal{C}^n$ of concepts $c_S : [0, 2^n] \to \{0,1\}$ such that

$$c_S(\mathbf{x}) = \begin{cases} 1, & \text{if } (\langle[\mathbf{x}]\rangle_b \text{ XOR } S) \quad \text{is odd.} \\ 0 & \text{o.w.} \end{cases} \tag{D.5}$$

where $[\cdot] : \mathbb{R} \to \mathbb{Z}$ rounds a decimal to its nearest integer, $\langle\cdot\rangle_b : \{0,\cdots,2^n\} \to \{0,1\}^n$ returns the binary encoding of the integer, and $(\langle[\mathbf{x}]\rangle_b \text{ XOR } S) = \sum_{j\in S}\langle[x]\rangle_b[j] \mod 2$. $\langle[x]\rangle_b[j]$ is the j^{th} least significant bit in the binary encoding of the nearest integer to $\mathbf{x}$. It is essentially the class of parity functions defined on the bits corresponding to the indices in S for the binary encoding of the nearest integer to $\mathbf{x}$. For example, as in Figure 5.6(a) if $S = \{0, 2\}$, then only the least significant and the third least significant bit of i are examined and the class label is 1 if an odd number of them are 1 and 0 otherwise.

Concept Class $\mathcal{H}$: Finally, we define the concept class $\mathcal{H} = \bigcup_{k=1}^\infty \mathcal{H}_k$ where $\mathcal{H}_k$ is the class of union of k intervals on the real line $\mathcal{H}^k$. Each concept $h_I \in \mathcal{H}^k$ can be written as a set of k disjoint intervals $I = \{I_1,\cdots,I_k\}$ on the real line i.e. for $1 \le j \le k$, $I_j = [a, b]$ where $0 \le a \le b$ and

$$h_I(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \bigcup_j I_j \\ 0 & \text{o.w.} \end{cases} \tag{D.6}$$

Now, we look at the algorithms to learn the concepts from $\mathcal{C}$ and $\mathcal{H}$ that minimise

the train error. Both of the algorithms will use a majority vote to determine the correct (de-noised) label for each interval, which will be necessary to minimise the test error. The intuition is that if we draw a sufficiently large number of samples, then the majority of samples on each interval will have the correct label with a high probability.

Lemma 8 proves that there exists an algorithm $\mathcal{A}$ such that $\mathcal{A}$ draws $m = \mathcal{O}\left(|\zeta|^2 \frac{(1-\eta)}{(1-2\eta)^2} \log \frac{|\zeta|}{\delta}\right)$ samples from the noisy oracle $\mathrm{EX}^{\eta}\left(\mathcal{D}_{s,\zeta}\right)$ and with probability $1-\delta$ where the probability is over the randomisation in the oracle, returns $f \in \mathcal{C}$ such that $\mathcal{R}(f; \mathcal{D}_{S,\zeta}) = 0$ and $\mathcal{R}_{\mathrm{Adv},\gamma}(f; \mathcal{D}_{S,\zeta}) = 0$ for all $\gamma < \frac{1}{4}$. As Lemma 8 states, the algorithm involves gaussian elimination over $|\zeta|$ variables and $|\zeta|$ majority votes (one in each interval) involving a total of m samples. Thus the algorithm runs in $\mathcal{O}\left(\mathrm{poly}\left(m\right) + \mathrm{poly}\left(|\zeta|\right)\right)$ time. Replacing the complexity of m and the fact that $|\zeta| = \mathcal{O}\left(n\right)$, the complexity of the algorithm is $\mathcal{O}\left(\mathrm{poly}\left(n, \frac{1}{1-2\eta}, \frac{1}{\delta}\right)\right)$.

Lemma 9 proves that there exists an algorithm $\widetilde{A}$ such that $\widetilde{A}$ draws

$$m > \max\left\{2\left|\zeta\right|^2 \log \frac{2\left|\zeta\right|}{\delta}\left(8\frac{(1-\eta)}{(1-2\eta)^2}+1\right), \frac{0.1\left|\zeta\right|}{\eta\gamma^2}\log\left(\frac{0.1\left|\zeta\right|}{\gamma\delta}\right)\right\}$$

samples and returns $h \in \mathcal{H}$ such that h has 0 training error, 0 test error and an adversarial test error of at least 0.1. We can replace $|\zeta| = \mathcal{O}\left(n\right)$ to get the required bound on m in the theorem. The algorithm to construct h visits every point at most twice - once during the construction of the intervals using majority voting, and once while accommodating for the mislabelled points. Replacing the complexity of m, the complexity of the algorithm is $\mathcal{O}\left(\mathrm{poly}\left(n, \frac{1}{1-2\eta}, \frac{1}{\gamma}, \frac{1}{\delta}\right)\right)$. This completes the proof. □

Lemma 8 (Parity Concept Class)**.** *There exists a learning algorithm $\mathcal{A}$ such that given access to the noisy example oracle $\mathrm{EX}^{\eta}\left(\mathcal{D}_{S,\zeta}\right)$, $\mathcal{A}$ makes $m = \mathcal{O}\left(|\zeta|^2 \frac{(1-\eta)}{(1-2\eta)^2} \log \frac{|\zeta|}{\delta}\right)$ calls to the oracle and returns a hypothesis $f \in \mathcal{C}$ such that with probability $1-\delta$, we have that $\mathcal{R}(f; \mathcal{D}_{S,\zeta}) = 0$ and $\mathcal{R}_{\mathrm{Adv},\gamma}(f; \mathcal{D}_{S,\zeta}) = 0$ for all $\gamma < \frac{1}{4}$.*

Proof. The algorithm $\mathcal{A}$ works as follows. It makes m calls to the oracle $\mathrm{EX}\left(\mathcal{D}_s^m\right)$ to obtain a set of points $\{(x_1, y_1), \cdots, (x_m, y_m)\}$ where $m \geq 2\left|\zeta\right|^2 \log \frac{2|\zeta|}{\delta}\left(8\frac{(1-\eta)}{(1-2\eta)^2}+1\right)$. Then, it replaces each x_i with $[x_i]$ ($[\cdot]$ rounds a decimal to the nearest integer) and then removes duplicate x_is by preserving the most frequent label y_i associated with each x_i. For example, if $\mathcal{S}_5 = \{(2.8, 1), (2.9, 0), (3.1, 1), (3.2, 1), (3.9, 0)\}$ then after this operation, we will have $\{(3, 1), (4, 0)\}$.

As $m \geq 2\left|\zeta\right|^2 \log \frac{2|\zeta|}{\delta}\left(8\frac{(1-\eta)}{(1-2\eta)^2}+1\right)$, using $\delta_2 = \frac{\delta}{2}$ and $k = \frac{8(1-\eta)}{(1-2\eta)^2}\log\frac{2|\zeta|}{\delta}$ in Lemma 10 guarantees that with probability $1-\frac{\delta}{2}$, each interval will have at least $\frac{8(1-\eta)}{(1-2\eta)^2}\log\frac{2|\zeta|}{\delta}$ samples.

Then for any specific interval, using $\delta_1 = \frac{2|\zeta|}{\delta}$ in Lemma 11 guarantees that with probability at least $1-\frac{2|\zeta|}{\delta}$, the majority vote for the label in that interval will

succeed in returning the de-noised label. Applying a union bound (Inequality 2) over all $|\zeta|$ intervals, will guarantee that with probability at least $1-\delta$, the majority label of every interval will be the denoised label.

Now, the problem reduces to solving a parity problem on this reduced dataset of $|\zeta|$ points (after denoising, all points in that interval can be reduced to the integer in the interval and the denoised label). We know that there exists a polynomial algorithm using Gaussian Elimination that finds a consistent hypothesis for this problem. We have already guaranteed that there is a point in $\mathcal{S}_m$ from every interval in the support of $\mathcal{D}_{S,\zeta}$. Further, f is consistent on $\mathcal{S}_m$ and f is constant in each of these intervals by design. Thus, with probability at least $1-\delta$ we have that $\mathcal{R}(f;\mathcal{D}_{S,\zeta})=0$.

By construction, f makes a constant prediction on each interval $\left(j-\frac{1}{2}, j+\frac{1}{2}\right)$ for all $j \in \zeta$. Thus, for any perturbation radius $\gamma < \frac{1}{4}$ the adversarial risk $\mathcal{R}_{\mathrm{Adv},\mathcal{D}_{S,\prime\zeta}}(f)=0$. Combining everything, we have shown that there is an algorithm that makes $2|\zeta|^2 \log\frac{2|\zeta|}{\delta}\left(8\frac{(1-\eta)}{(1-2\eta)^2}+1\right)$ calls to the $\mathrm{EX}\left(\mathcal{D}^{\eta}_{S,\zeta}\right)$ oracle, runs in time polynomial in $|\zeta|, \frac{1}{1-2\eta}, \frac{1}{\delta}$ to return $f \in \mathcal{C}$ such that $\mathcal{R}(f;\mathcal{D}_{S,\zeta})=0$ and $\mathcal{R}_{\mathrm{Adv},\gamma}(f;\mathcal{D}_{S,\zeta})=0$ for $\gamma<\frac{1}{4}$. □

Lemma 9 (Union of Interval Concept Class)**.** *There exists a learning algorithm $\widetilde{\mathcal{A}}$ such that given access to a noisy example oracle makes $m=\mathcal{O}\left(|\zeta|^2\frac{(1-\eta)}{(1-2\eta)^2}\log\frac{|\zeta|}{\delta}\right)$ calls to the oracle and returns a hypothesis $h\in\mathcal{H}$ such that training error is 0 and with probability $1-\delta$, $\mathcal{R}(f;\mathcal{D}_{S,\zeta})=0$.*

Further for any $h\in\mathcal{H}$ that has zero training error on m' samples drawn from $\mathrm{EX}^{\eta}(\mathcal{D}_{S,\zeta})$ for $m'>\frac{|\zeta|}{10\eta\gamma^2}\log\frac{|\zeta|}{10\gamma\delta}$ and $\eta\in\left(0,\frac{1}{2}\right)$ then $\mathcal{R}_{\mathrm{Adv},\gamma}(f;\mathcal{D}_{S,\zeta})\geq 0.1$ for all $\gamma>0$.

Proof of Lemma 9. The first part of the algorithm works similarly to Lemma 8. The algorithm $\widetilde{\mathcal{A}}$ makes m calls to the oracle $\mathrm{EX}(\mathcal{D}^m_s)$ to obtain a set of points $\mathcal{S}_m=\{(x_1,y_1),\cdots,(x_m,y_m)\}$ where $m\geq 2|\zeta|^2\log\frac{2|\zeta|}{\delta}\left(8\frac{(1-\eta)}{(1-2\eta)^2}+1\right)$. $\widetilde{\mathcal{A}}$ computes $h\in\mathcal{H}$ as follows. To begin, let the list of intervals in h be I and $\mathcal{M}_z=\{\}$ Then do the following for every $(x,y)\in\mathcal{S}_m$.

1. let $z:=[x]$,
2. Let $\mathcal{N}_z\subseteq\mathcal{S}_m$ be the set of all $(x,y)\in\mathcal{S}_m$ such that $|x-z|<0.5$.
3. Compute the majority label $\tilde{y}$ of $\mathcal{N}_z$.
4. Add all $(x,y)\in\mathcal{N}_z$ such that $y\neq\tilde{y}$ to $\mathcal{M}_z$
5. If $\tilde{y}=1$, then add the interval $(z-0.5,z+0.5)$ to I.
6. Remove all elements of $\mathcal{N}_z$ from $\mathcal{S}_m$ i.e. $\mathcal{S}_m:=\mathcal{S}_m\setminus\mathcal{N}_z$.

For reasons similar to Lemma 8, as $m\geq 2|\zeta|^2\log\frac{2|\zeta|}{\delta}\left(8\frac{(1-\eta)}{(1-2\eta)^2}+1\right)$, Lemma 10 guarantees that with probability $1-\frac{\delta}{2}$, each interval will have at least $\frac{8(1-\eta)}{(1-2\eta)^2}\log\frac{2|\zeta|}{\delta}$

samples. Then for any specific interval, Lemma 11 guarantees that with probability at least $1 - \frac{2|\zeta|}{\delta}$, the majority vote for the label in that interval will succeed in returning the de-noised label. Applying a union bound (Inequality 2) over all intervals, will guarantee that with probability at least $1 - \delta$, the majority label of every interval will be the denoised label. As each interval inζ has at least one point, all the intervals in ζ with label 1 will be included in I with probability $1 - \delta$. Thus, $\mathcal{R}(h; \mathcal{D}_{S,\zeta}) = 0$.

Now, for all $(x, y) \in \mathcal{M}_z$, add the interval $[x]$ to I if $y = 1$. If $y = 0$ then x must lie a interval $(a, b) \in I$. Replace that interval as follows $I := I \backslash (a, b) \cup \{(a, x), (x, b)\}$. As only a finite number of sets with Lebesgue measure of 0 were added or deleted from I, the net test error of h doesn't change and is still 0 i.e. $\mathcal{R}(h; \mathcal{D}_{S,\zeta}) = 0$

For the second part, we will invoke Theorem 2. To avoid confusion in notation, we will use Γ instead of ζ to refer to the sets in Theorem 2 and reserve ζ for the support of interval of $\mathcal{D}_{S,\zeta}$. Let Γ be any set of disjoint intervals of width $\frac{\gamma}{2}$ such that $|\Gamma| = \frac{0.1|\zeta|}{\gamma}$. This is always possible as the total width of all intervals in Γ is $\frac{0.1|\zeta|}{\gamma}\frac{\gamma}{2} = 0.1\frac{|\zeta|}{2}$ which is less than the total width of the support $\frac{|\zeta|}{2}$. c_1, c_2 from Eq. Equation (5.2) is

$$c_1 = \mathbb{P}_{\mathcal{D}_{S,\zeta}}[\Gamma] = \frac{2 * 0.1\,|\zeta|}{2\,|\zeta|} = 0.1, \quad c_2 = \frac{2\gamma}{2\,|\zeta|}\,|\zeta| = \gamma$$

Thus, if h has an error of zero on a set of m' examples drawn from $\mathrm{EX}^{\eta}(\mathcal{D}_{S,\zeta})$ where $m' > \frac{0.1|\zeta|}{\eta\gamma^2}\log\left(\frac{0.1|\zeta|}{\gamma\delta}\right)$, then by Theorem 2, $\mathcal{R}_{\mathrm{Adv},\gamma}(h; \mathcal{D}_{S,\zeta}) > 0.1$.

Combining the two parts for

$$m > \max\left\{2\,|\zeta|^2 \log\frac{2\,|\zeta|}{\delta}\left(8\frac{(1-\eta)}{(1-2\eta)^2} + 1\right), \frac{0.1\,|\zeta|}{\eta\gamma^2}\log\left(\frac{0.1\,|\zeta|}{\gamma\delta}\right)\right\}$$

it is possible to obtain $h \in \mathcal{H}$ such that h has zero training error, $\mathcal{R}(\mathcal{D}_{S,\zeta}; h) = 0$ and $\mathcal{R}_{\mathrm{Adv},\gamma}(h; \mathcal{D}_{S,\zeta}) > 0.1$ for any $\gamma > 0$.

□

Lemma 10. *Given $k \in \mathbb{Z}_+$ and a distribution $\mathcal{D}_{S,\zeta}$, for any $\delta_2 > 0$ if $m > 2\,|\zeta|^2\,k + 2\,|\zeta|^2 \log\frac{|\zeta|}{\delta_2}$ samples are drawn from $\mathrm{EX}(\mathcal{D}_{S,\zeta})$ then with probability at least $1 - \delta_2$ there are at least k samples in each interval $\left(j - \frac{1}{4}, j + \frac{1}{4}\right)$ for all $j \in \zeta$.*

Proof of Lemma 10. We will repeat the following procedure $|\zeta|$ times once for each interval in ζ and show that with probability $\frac{\delta}{|\zeta|}$ the j^{th} run will result in at least k samples in the j^{th} interval.

Corresponding to each interval in ζ, we will sample at least m' samples where $m' = 2\,|\zeta|\,k + 2\,|\zeta| \log\frac{|\zeta|}{\delta_2}$. If z_i^j is the random variable that is 1 when the i^{th} sample belongs to the j^{th} interval, then j^{th} interval has at least k points out of the m' points

sampled for that interval with probability less than $\frac{\delta_2}{|\zeta|}$.

$$\begin{aligned}
\mathbb{P}\left[\sum_i z_i^j \leq k\right] &= \mathbb{P}\left[\sum_i z_i^j \leq (1-\delta)\,\mu\right] && \delta = 1 - \frac{k}{\mu}, \mu = \mathbb{E}\left[\sum_i z_i^j\right] \\
&\leq \exp\left(-\left(1-\frac{k}{\mu}\right)^2 \frac{\mu}{2}\right) && \text{By Chernoff's inequality (Inequality 6)} \\
&\leq \exp\left(-\left(\frac{m'}{2\,|\zeta|} - k + \frac{k^2\,|\zeta|}{2m'}\right)\right) && \mu = \frac{m'}{|\zeta|} \\
&\leq \exp\left(k - \frac{m'}{2\,|\zeta|}\right) \leq \frac{\delta_2}{|\zeta|}
\end{aligned}$$

where the last step follows from $m' > 2\,|\zeta|\,k + 2\,|\zeta|\log\frac{|\zeta|}{\delta_2}$. With probability at least $\frac{\delta}{|\zeta|}$, every interval will have at least k samples. Finally, an union bound (Inequality 2) over each interval gives the desired result. As we repeat the process for all $|\zeta|$ intervals, the total number of samples drawn will be at least $|\zeta|\,m' = 2\,|\zeta|^2\,k + 2\,|\zeta|^2\log\frac{|\zeta|}{\delta_2}$. □

Lemma 11 (Majority Vote)**.** *For a given $y \in \{0,1\}$, let $S = \{s_1, \cdots, s_m\}$ be a set of size m where each element is y with probability $1-\eta$ and $1-y$ otherwise. If $m > \frac{8(1-\eta)}{(1-2\eta)^2}\log\frac{1}{\delta_1}$ then with probability at least $1-\delta_1$ the majority of S is y.*

Proof of Lemma 11. Without loss of generality let $y = 1$. For the majority to be 1 we need to show that there are more than $\frac{m}{2}$ "1"s in S i.e. we need to show that the following probability is less than δ_1.

$$\begin{aligned}
\mathbb{P}\left[\sum s_i < \frac{m}{2}\right] &= \mathbb{P}\left[\sum s_i < \frac{m}{2\mu} * \mu + \mu - \mu\right] && \mu = \mathbb{E}\left[\sum s_i\right] \\
&= \mathbb{P}\left[\sum s_i < \left(1 - \left(1 - \frac{m}{2\mu}\right)\right)\mu\right] \\
&\leq \exp\left(-\frac{(1-2\eta)^2}{8\,(1-\eta)^2}\mu\right) && \text{By Chernoff's Inequality (Inequality 6)} \\
&= \exp\left(-\frac{(1-2\eta)^2}{8\,(1-\eta)}m\right) && \because \mu = (1-\eta)\,m \\
&\leq \delta_1 && \because m > \frac{8\,(1-\eta)}{(1-2\eta)^2}\log\frac{1}{\delta_1}
\end{aligned}$$

□

E. Appendix for Chapter 6

E.1 Proofs for Section 6.3.2

Proof of Lemma 3. By the Construction of the Nyström SVD algorithm, we know that $vecX_r = \mathbf{C}\mathbf{W}_r^{+}\mathbf{C}^T$. We will first show that $\mathbf{W}_r^{+}$ is a symmetric matrix.

We know that $\mathbf{X}$ is SPSD. Let $\mathbf{I}$ be a sorted list of distinct indices of length l. Then by construction of $\mathbf{W}$,

$$\mathbf{W}_{i,j} = \mathbf{X}_{I[i],\mathbf{I}[j]}$$

As $\mathbf{X}_{\mathbf{I}[i],\mathbf{I}[j]} = \mathbf{X}_{\mathbf{I}[j],\mathbf{I}[i]}$, $\mathbf{W}$ is symmetric. At this step, our algorithm adds $\delta \cdot \mathbf{I}$ to $\mathbf{W}$ where $\delta \geq 0$. We show that $W + \delta \cdot \mathcal{I}$ is positive semidefinite. Consider a vector $\mathbf{a} \in \mathbb{R}^{|X|}$. Create a vector $\bar{\mathbf{a}} \in \mathbb{R}^m$ where

$$\bar{\mathbf{a}}_i = \begin{cases} 0 & \text{if } i \notin I \\ \mathbf{a}_i & \text{o.w.} \end{cases}$$

$$\mathbf{a}^\top (\mathbf{W} + \delta \cdot \mathcal{I})\, \mathbf{a} = \bar{\mathbf{a}}^\top X \bar{\mathbf{a}} + \delta \cdot \mathbf{a}^\top \mathbf{I}\mathbf{a} \geq 0 + \delta \|\mathbf{a}\|^2 \geq 0 \tag{E.1}$$

Let $\mathbf{W} + \delta\mathbf{I}$ be the new $\mathbf{W}$; Equation (E.1) shows that the updated $\mathbf{W}$ is also positive semidefinite.

Now we will show that $\mathbf{X}_r$ is symmetric as well. As $\mathbf{W}$ is symmetric, there exists an orthogonal matrix $\mathbf{Q}$ and a non-negative diagonal matrix Λ such that

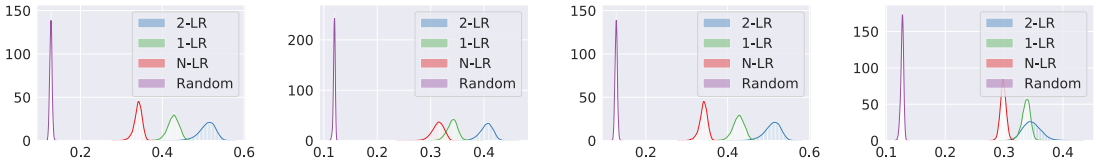
$$W = Q\Lambda Q^T$$

We know that $\mathbf{W}_r = \mathbf{Q}_{[1:r]}\Lambda_{[1:r]}\mathbf{Q}_{[1:r]}^T$ and $\mathbf{W}_r^{+} = \mathbf{Q}_{[1:r]}\Lambda_{[1:r]}^{-1}\mathbf{Q}_{[1:r]}^T$.
Hence,

$$\begin{aligned} \mathbf{X}_r &= \mathbf{C}\mathbf{W}_r^{+}\mathbf{C}^T \\ &= \mathbf{C}\mathbf{Q}_{[1:r]}\Lambda_{[1:r]}^{-1}\mathbf{Q}_{[1:r]}^T\mathbf{C}^T \\ \mathbf{X}_r^T &= (\mathbf{C}\mathbf{Q}_{[1:r]}\Lambda_{[1:r]}^{-1}\mathbf{Q}_{[1:r]}^T\mathbf{C}^T)^T \\ &= \mathbf{C}\mathbf{Q}_{[1:r]}\Lambda_{[1:r]}^{-1}\mathbf{Q}_{[1:r]}^T\mathbf{C}^T \\ &= \mathbf{X}_r \end{aligned}$$

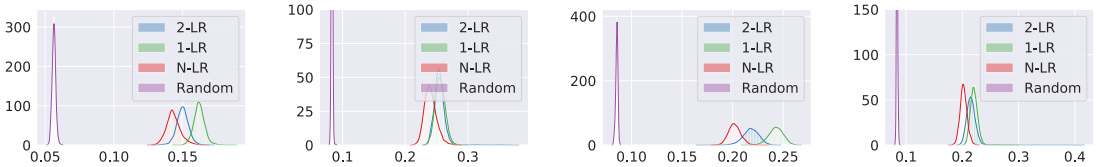
$\therefore \mathbf{X}_r$ is symmetric. We can also see that the $\mathbf{X}_r^T$ is positive semi definite by pre-multiplying and post multiplying it with a non-zero vector and using the fact that $\mathbf{W}_r^{+}$ is positive semi-definite. □

E.2 Additional Results



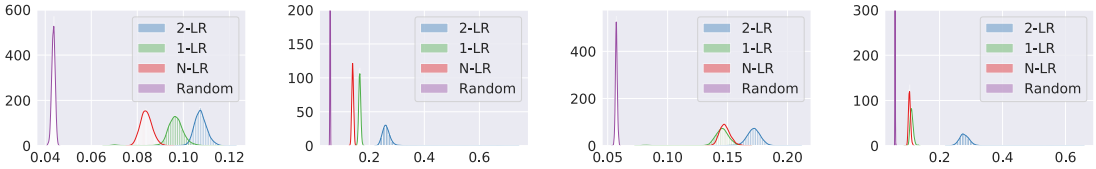
Conv Layer 1 Conv Layer 2 Conv Layer 3 Conv Layer 4

(a) Cushion of ResNet-18 Block 1 on CIFAR10



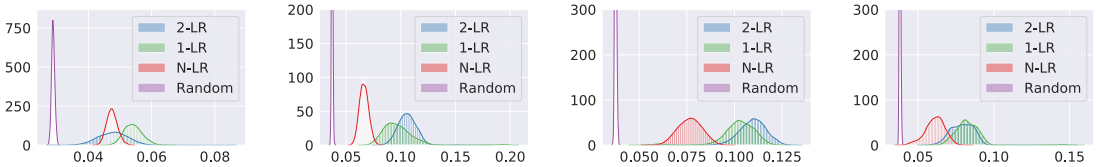
Conv Layer 1 Conv Layer 2 Conv Layer 3 Conv Layer 4

(b) Cushion of ResNet-18 Block 2 on CIFAR10



Conv Layer 1 Conv Layer 2 Conv Layer 3 Conv Layer 4

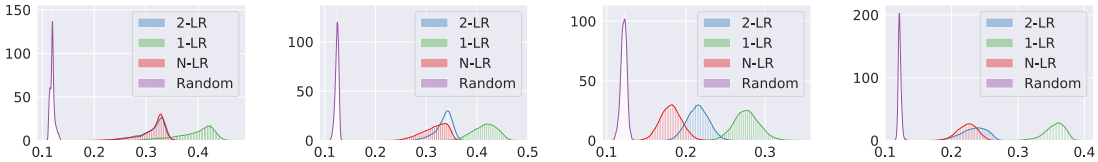
(c) Cushion of ResNet-18 Block 3 on CIFAR10



Conv Layer 1 Conv Layer 2 Conv Layer 3 Conv Layer 4

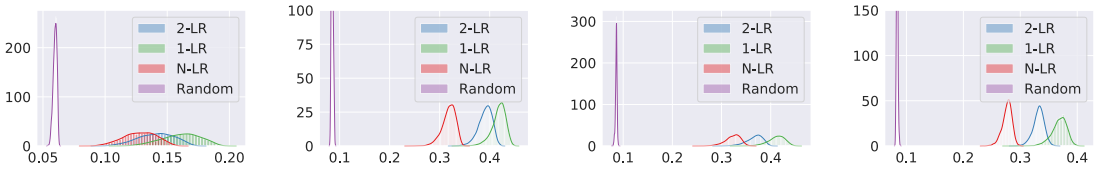
(d) Cushion of ResNet-18 Block 4 on CIFAR10.

Figure E.1: Layer Cushion of various layers of a ResNet-18 trained on CIFAR10. Each ResNet-18 has four blocks with each block containing four convolutional layers.



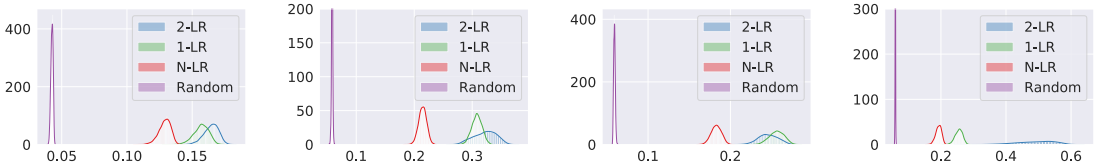
Conv Layer 1 Conv Layer 2 Conv Layer 3 Conv Layer 4

(a) Cushion of ResNet-18 Block 1 on SVHN.



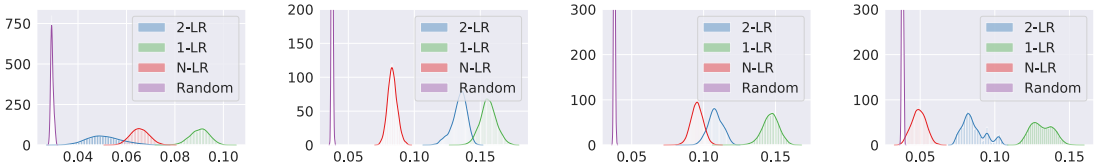
Conv Layer 1 Conv Layer 2 Conv Layer 3 Conv Layer 4

(b) Cushion of ResNet-18 Block 2 on SVHN.



Conv Layer 1 Conv Layer 2 Conv Layer 3 Conv Layer 4

(c) Cushion of ResNet-18 Block 3 on SVHN.



Conv Layer 1 Conv Layer 2 Conv Layer 3 Conv Layer 4

(d) Cushion of ResNet-18 Block 4 on SVHN.

Figure E.2: Layer Cushion of various layers of a ResNet-19 trained on SVHN. Each ResNet-18 has four blocks with each block containing four convolutional layers.

F. Appendix for Chapter 7

F.1 Proofs for Section 7.3

Proof of Theorem 5. let K denotes the number of classes, $\mathcal{L}_f$ denote the focal loss with parameter γ, $\mathcal{L}_c$ denote the cross entropy between $\hat{p}$ and q, and let q_y denotes the ground-truth probability assigned to the y^{th} class (similarly for $\hat{p}_y$). We consider the following simple extension of focal loss:

$$
\begin{aligned}
\mathcal{L}_f &= -\sum_{y=1}^{K}(1-\hat{p}_y)^{\gamma} q_y \log \hat{p}_y \\
&\geq -\sum_{y=1}^{K}(1-\gamma\hat{p}_y) q_y \log \hat{p}_y && \text{By Bernoulli's inequality } \forall\gamma \geq 1, \text{ since } \hat{p}_y \in [0,1] \\
&= -\sum_{y=1}^{K} q_y \log \hat{p}_y - \gamma \left| \sum_{y=1}^{K} q_y \hat{p}_y \log \hat{p}_y \right| && \forall y,\ \log \hat{p}_y \leq 0 \\
&\geq -\sum_{y=1}^{K} q_y \log \hat{p}_y - \gamma \max_j q_j \sum_{y=1}^{K} |\hat{p}_y \log \hat{p}_y| && \text{By Hölder's inequality } ||fg||_1 \leq ||f||_\infty ||g||_1 \\
&\geq -\sum_{y=1}^{K} q_y \log \hat{p}_y + \gamma \sum_{y=1}^{K} \hat{p}_y \log \hat{p}_y && \forall j, q_j \in [0,1] \\
&= \mathcal{L}_c - \gamma \mathbb{H}[\hat{p}].
\end{aligned}
$$

We know that $\mathcal{L}_c = \mathrm{KL}(q||\hat{p}) + \mathbb{H}[q]$. Combining this equality with the above inequality leads to:

$$
\mathcal{L}_f \geq \mathrm{KL}(q||\hat{p}) + \underbrace{\mathbb{H}[q]}_{constant} - \gamma \mathbb{H}[\hat{p}].
$$

□

Here we provide the proofs of both the Lemmas presented in the main text. While Theorem 6 helps us understand the regularisation effect of focal loss, Theorem 7 provides the γ values in a principled way such that it is sample-dependent. Implementing the sample-dependent γ is very easy as implementation of the Lambert-W function [58] is available in standard libraries (e.g. python scipy).

Proof of Theorem 6. Let $\mathbf{w}$ be the linear layer parameters connecting the feature map to the logit s. Then, using the chain rule, $\frac{\partial \mathcal{L}_f}{\partial \mathbf{w}} = \left(\frac{\partial s}{\partial \mathbf{w}}\right)\left(\frac{\partial \hat{p}_{i,y_i}}{\partial s}\right)\left(\frac{\partial \mathcal{L}_f}{\partial \hat{p}_{i,y_i}}\right)$. Similarly, $\frac{\partial \mathcal{L}_c}{\partial \mathbf{w}} = \left(\frac{\partial s}{\partial \mathbf{w}}\right)\left(\frac{\partial \hat{p}_{i,y_i}}{\partial s}\right)\left(\frac{\partial \mathcal{L}_c}{\partial \hat{p}_{i,y_i}}\right)$. The derivative of the focal loss with respect to

$\hat{p}_{i,y_i}$, the softmax output of the network for the true class y_i, takes the form

$$\begin{aligned}\frac{\partial \mathcal{L}_f}{\partial \hat{p}_{i,y_i}} &= -\frac{1}{\hat{p}_{i,y_i}}\Big((1-\hat{p}_{i,y_i})^{\gamma} - \gamma \hat{p}_{i,y_i}(1-\hat{p}_{i,y_i})^{\gamma-1}\log(\hat{p}_{i,y_i})\Big) \\ &= \frac{\partial \mathcal{L}_c}{\partial \hat{p}_{i,y_i}} g(\hat{p}_{i,y_i},\gamma),\end{aligned}$$

in which $g(\hat{p}_{i,y_i},\gamma) = (1-\hat{p}_{i,y_i})^{\gamma} - \gamma \hat{p}_{i,y_i}(1-\hat{p}_{i,y_i})^{\gamma-1}\log(\hat{p}_{i,y_i})$ and $\frac{\partial \mathcal{L}_c}{\partial \hat{p}_{i,y_i}} = -\frac{1}{\hat{p}_{i,y_i}}$. It is thus straightforward to verify that if $g(\hat{p}_{i,y_i},\gamma) \in [0,1]$, then $\left\|\frac{\partial \mathcal{L}_{\mathrm{f}}}{\partial \hat{\mathrm{p}}_{\mathrm{i,y_i}}}\right\| \leq \left\|\frac{\partial \mathcal{L}_{\mathrm{c}}}{\partial \hat{\mathrm{p}}_{\mathrm{i,y_i}}}\right\|$, which itself implies that $\left\|\frac{\partial \mathcal{L}_{\mathrm{f}}}{\partial \mathbf{w}}\right\| \leq \left\|\frac{\partial \mathcal{L}_{\mathrm{c}}}{\partial \mathbf{w}}\right\|$. □

Proof of Theorem 7. We derive the value of $\gamma > 0$ for which $g(p_0,\gamma) = 1$ for a given $p_0 \in [0,1]$. From Proposition 4.1, we already know that

$$\frac{\partial \mathcal{L}_f}{\partial \hat{p}_{i,y_i}} = \frac{\partial \mathcal{L}_c}{\partial \hat{p}_{i,y_i}} g(\hat{p}_{i,y_i},\gamma), \tag{F.1}$$

where $\mathcal{L}_f$ is focal loss, $\mathcal{L}_c$ is cross entropy loss, $\hat{p}_{i,y_i}$ is the probability assigned by the model to the ground-truth correct class for the i^{th} sample, and

$$g(\hat{p}_{i,y_i},\gamma) = (1-\hat{p}_{i,y_i})^{\gamma} - \gamma \hat{p}_{i,y_i}(1-\hat{p}_{i,y_i})^{\gamma-1}\log(\hat{p}_{i,y_i}). \tag{F.2}$$

For $p \in [0,1]$, if we look at the function $g(p,\gamma)$, then we can clearly see that $g(p,\gamma) \to 1$ as $p \to 0$, and that $g(p,\gamma) = 0$ when $p = 1$. To observe the behaviour of $g(p,\gamma)$ for intermediate values of p, we first take its derivative with respect to p:

$$\frac{\partial g(p,\gamma)}{\partial p} = \gamma(1-p)^{\gamma-2}\big[-2(1-p) - (1-p)\log p + (\gamma-1)p\log p\big] \tag{F.3}$$

In Equation F.3, $\gamma(1-p)^{\gamma-2} > 0$ except when $p = 1$ (in which case $\gamma(1-p)^{\gamma-2} = 0$). Thus, to observe the sign of the gradient $\frac{\partial g(p,\gamma)}{\partial p}$, we focus on the term

$$-2(1-p) - (1-p)\log p + (\gamma-1)p\log p. \tag{F.4}$$

Dividing Equation F.4 by $(-\log p)$, the sign remains unchanged and we get

$$k(p,\gamma) = \frac{2(1-p)}{\log p} + 1 - \gamma p. \tag{F.5}$$

We can see that $k(p,\gamma) \to 1$ as $p \to 0$ and $k(p,\gamma) \to -(1+\gamma)$ as $p \to 1$ (using l'Hôpital's rule). Furthermore, $k(p,\gamma)$ is monotonically decreasing for $p \in [0,1]$. Thus, as the gradient $\frac{\partial g(p,\gamma)}{\partial p}$ is positive initially starting from $p = 0$ and negative later till $p = 1$, we can say that $g(p,\gamma)$ first monotonically increases starting from 1 (as $p \to 0$) and then monotonically decreases down to 0 (at $p = 1$). Thus, if for some threshold $p_0 > 0$ and for some $\gamma > 0$, $g(p,\gamma) = 1$, then $\forall p > p_0$, $g(p,\gamma) < 1$. We now want to find a γ such that $\forall p \geq p_0$, $g(p,\gamma) \leq 1$. First, let $a = (1-p_0)$ and

$b = p_0 \log p_0$. Then:

$$
\begin{aligned}
& g(p_0, \gamma) = (1-p_0)^{\gamma} - \gamma p_0 (1-p_0)^{\gamma-1} \log p_0 \leq 1 \\
\implies & (1-p_0)^{\gamma-1}[(1-p_0) - \gamma p_0 \log p_0] \leq 1 \\
\implies & a^{\gamma-1}(a - \gamma b) \leq 1 \\
\implies & (\gamma - 1)\log a + \log(a - \gamma b) \leq 0 \\
\implies & \Big(\gamma - \frac{a}{b}\Big)\log a + \log(a - \gamma b) \leq \Big(1 - \frac{a}{b}\Big)\log a \\
\implies & (a - \gamma b) e^{(\gamma - a/b)\log a} \leq a^{(1-a/b)} \\
\implies & \Big(\gamma - \frac{a}{b}\Big) e^{(\gamma - a/b)\log a} \leq -\frac{a^{(1-a/b)}}{b} \\
\implies & \Big(\Big(\gamma - \frac{a}{b}\Big)\log a\Big) e^{(\gamma - a/b)\log a} \geq -\frac{a^{(1-a/b)}}{b}\log a
\end{aligned}
\tag{F.6}
$$

where $a = (1-p_0)$ and $b = p_0 \log p_0$. We know that the inverse of $y = xe^x$ is defined as $x = W(y)$, where W is the Lambert-W function [58]. Furthermore, the r.h.s. of the inequality in Equation F.6 is always negative, with a minimum possible value of $-1/e$ that occurs at $p_0 = 0.5$. Therefore, applying the Lambert-W function to the r.h.s. will yield two real solutions (corresponding to a principal branch denoted by W_0 and a negative branch denoted by W_{-1}). We first consider the solution corresponding to the negative branch (which is the smaller of the two solutions):

$$
\begin{aligned}
& \Big((\gamma - \frac{a}{b})\log a\Big) \leq W_{-1}\Big(-\frac{a^{(1-a/b)}}{b}\log a\Big) \\
\implies & \gamma \geq \frac{a}{b} + \frac{1}{\log a} W_{-1}\Big(-\frac{a^{(1-a/b)}}{b}\log a\Big)
\end{aligned}
\tag{F.7}
$$

If we consider the principal branch, the solution is

$$
\gamma \leq \frac{a}{b} + \frac{1}{\log a} W_0\Big(-\frac{a^{(1-a/b)}}{b}\log a\Big), \tag{F.8}
$$

which yields a negative value for γ that we discard. Thus Equation F.7 gives the values of γ for which if $p > p_0$, then $g(p, \gamma) < 1$. In other words, $g(p_0, \gamma) = 1$, and for any $p < p_0$, $g(p, \gamma) > 1$. □

Bibliography

[1] Microsoft translator. https://www.microsoft.com/en-us/translator/. Accessed: 2020-05-10. 1

[2] Gm cruise. https://www.getcruise.com/technology. Accessed: 2020-05-10. 1

[3] Lengoo: Neural machine translation for enterprise use. https://www.lengoo.com/ibb/. Accessed: 2020-05-10. 1

[4] Pixsy image search. https://www.pixsy.com/monitor/. Accessed: 2020-05-10. 1

[5] Salesforce einstein vision. https://metamind.readme.io/docs/what-is-the-predictive-vision-service. Accessed: 2020-05-10. 1

[6] Tesla autopilot ai. https://www.tesla.com/en_GB/autopilotAI. Accessed: 2020-05-10. 1

[7] Tilde neural machine translation. https://www.tilde.com/products-and-services/machine-translation/neural-machine-translation. Accessed: 2020-05-10. 1

[8] Tineye wine engine. https://services.tineye.com/WineEngine. Accessed: 2020-05-10. 1

[9] Waymo driver. https://waymo.com/waymo-driver/. Accessed: 2020-05-10. 1

[10] Naveed Akhtar, Jian Liu, and Ajmal Mian. Defense against universal adversarial perturbations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 47

[11] Tiago Alves and Don Felton. Trustzone: Integrated hardware and software security. *White Paper*, 2004. 62

[12] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 1988. 12

[13] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, and Others. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security*, 2017. 167

[14] Diego Ardila, Atilla P. Kiraly, Sujeeth Bharadwaj, Bokyung Choi, Joshua J. Reicher, Lily Peng, Daniel Tse, Mozziyar Etemadi, Wenxing Ye, Greg Corrado, David P. Naidich, and Shravya Shetty. End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. *Nature Medicine*, 2019. 52

[15] Sanjeev Arora and Yi Zhang. Do gans learn the distribution? some theory and empirics. In *International Conference on Learning Representations (ICLR)*, 2018. 92

[16] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning (ICML)*, 2018. 25, 32, 33, 34, 75, 76, 139

[17] Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *International Conference on Learning Representations (ICLR)*, 2020. 60

[18] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning (ICML)*, 2018. 44, 95

[19] Pranjal Awasthi, Himanshu Jain, Ankit Singh Rawat, and Aravindan Vijayaraghavan. Adversarial robustness via robust low rank representations. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020. 49

[20] Eugene Bagdasaryan and Vitaly Shmatikov. Differential privacy has disparate impact on model accuracy. *arXiv:1905.12101*, 2019. 108

[21] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. 2002. 25, 28, 29

[22] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 25, 30, 31, 67, 88

[23] Peter L. Bartlett, Philip M. Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. 2020. 17, 96

[24] Kenneth E Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, 1968. 177

[25] Mikhail Belkin, Daniel J Hsu, and Partha Mitra. Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 96

[26] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. In *International Conference on Machine Learning (ICML)*, 2018. 16, 17, 96

[27] Mikhail Belkin, Daniel Hsu, and Ji Xu. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science*, 2019. 96

[28] Mikhail Belkin, Alexander Rakhlin, and Alexandre B. Tsybakov. Does data interpolation contradict statistical optimality? In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019. 96

[29] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 2018. 36, 95, 118, 132

[30] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 1995. 45

[31] Saul Blecker, David Sontag, Leora I Horwitz, Gilad Kuperman, Hannah Park, Alex Reyentovich, and Stuart D Katz. Early Identification of Patients With Acute Decompensated Heart Failure. *Journal of cardiac failure*, 2017. 23, 61

[32] Thomas Blumensath and Mike E Davies. Iterative hard thresholding for compressed sensing. *Applied and computational harmonic analysis*, 2009. 19

[33] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *International Cryptology Conference (CRYPTO)*, 2017. 167, 168, 170

[34] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research (JMLR)*, 2002. 14

[35] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*, 2014. 65

[36] Idar Johan Brekke, Lars Håland Puntervoll, Peter Bank Pedersen, John Kellett, and Mikkel Brabrand. The value of vital sign trends in predicting and monitoring clinical deterioration: A systematic review. *PLOS ONE*, 2019. 52

[37] Glenn W Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 1950. 56, 59, 161

[38] Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. 2016. 83

[39] Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya P. Razenshteyn. Adversarial examples from computational constraints. *arXiv:1811.06418*, 2018. 50, 183

[40] Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya P. Razenshteyn. Adversarial examples from computational constraints. In *International Conference on Machine Learning (ICML)*, 2019. 50

[41] Rudy Bunel, Ilker Turkaslan, Philip H.S. Torr, Pushmeet Kohli, and M. Pawan Kumar. A unified view of piecewise linear neural network verification. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 48

[42] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 2009. 19

[43] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*, 2017. 36, 38, 95, 118, 132

[44] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM Workshop on Artificial Intelligence and Security*, 2017. 47

[45] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017. 168, 170, 174

[46] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, and Peter Rindal. Private collaborative neural network learning. *IACR Cryptology ePrint Archive*, 2017. 168, 170

[47] Niladri S. Chatterji and Philip M. Long. Finite-sample analysis of interpolating linear classifiers in the overparameterized regime. *arXiv:2004.12019*, 2020. 17, 96

[48] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. 38

[49] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning (ICML)*, 2015. 174

[50] Xi Chen, Xiaolin Hu, Hucheng Zhou, and Ningyi Xu. Fxpnet: Training a deep convolutional neural network in fixed-point representation. In *International Joint Conference on Neural Networks (IJCNN)*, 2017. 174

[51] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 1

[52] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, 2017. 48

[53] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Applications of Cryptology and Information Security*, 2016. 65, 173

[54] Matthew M. Churpek and Dana P. Edelson. Moving beyond single-parameter early warning scores for rapid response system activation. *Critical Care Medicine*, 2016. 52

[55] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning (ICML)*, 2017. 19, 43, 44

[56] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning (ICML)*, 2019. 43, 48, 49

[57] Adrian Corduneanu and Tommi S. Jaakkola. On information regularization. In *Conference on Uncertainty in Artificial Intelligence (UAI*, 2002. 21

[58] Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. On the Lambert W Function. *Advances in Computational Mathematics*, 1996. 158, 216, 218

[59] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *USENIX Conference on Security Symposium*, 2016. 62

[60] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015. 174

[61] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. In *Conference on Neural*

Information Processing Systems (NeurIPS), 2016. vii, 65, 167, 173, 182, 190, 193

[62] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data mining* , 2004. 36, 95, 118, 132

[63] Pau de Jorge, Amartya Sanyal, Harkirat S. Behl, Philip H. S. Torr, Gregory Rogez, and Puneet K. Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. In *International Conference on Learning Representations (ICLR)*, 2021. viii

[64] Akshay Degwekar, Preetum Nakkiran, and Vinod Vaikuntanathan. Computational limitations in robust classification and win-win results. In *Conference on Learning Theory (COLT)*, 2019. 50, 51, 111, 183

[65] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via theEMAlgorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1977. 123

[66] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 161, 191

[67] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014. 145

[68] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=H1uR4GZRZ`. 45

[69] Zhipeng Ding, Xu Han, Peirong Liu, and Marc Niethammer. Local temperature scaling for probability calibration. *arXiv:2008.05105*, 2020. 58

[70] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *International Conference on Machine Learning (ICML)*, 2014. 118

[71] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`. 193

[72] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2015. 65, 172

[73] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *International Conference on Learning Representations (ICLR)*, 2017. 90

[74] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Conference on Uncertainty in Artificial Intelligence (UAI*, 2017. 25

[75] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1936. 78

[76] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, 2017. 48

[77] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017. 20

[78] Deng-Ping Fan, Ge-Peng Ji, Tao Zhou, Geng Chen, Huazhu Fu, Jianbing Shen, and Ling Shao. Pranet: Parallel reverse attention network for polyp segmentation. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2020. 1

[79] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. Adversarial vulnerability for any classifier. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 49

[80] Vitaly Feldman. Does learning require memorization? a short tale about a long tail. In *ACM Symposium on Theory of Computing (STOC)*, 2020. 105

[81] Santo Fortunato. Community detection in graphs. *Physics reports*, 2010. 23, 61

[82] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning (ICML)*, 2017. 1

[83] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard S. Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2020. 117

[84] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC)*, 2009. vii, 6, 65, 171, 172

[85] Craig Gentry, Shai Halevi, and Nigel P Smart. Fully homomorphic encryption with polylog overhead. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2012. 65

[86] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *International Cryptology Conference (CRYPTO)*, 2013. 65

[87] Sushmito Ghosh and Douglas L Reilly. Credit card fraud detection with a neural-network. In *International Conference on System Sciences*, 1994. 23, 61

[88] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *International Conference on Machine Learning (ICML)*, 2016. vii, 65, 168, 170, 172, 183

[89] Morgane Goibert and Elvis Dohmatob. Adversarial robustness via label-smoothing. *arXiv:1906.11567*, 2019. 46

[90] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *ACM Symposium on Theory of Computing (STOC)*, 1987. 63

[91] Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. In *Conference on Learning Theory (COLT)*, 2018. 28, 29, 31

[92] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014. 68, 70

[93] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations (ICLR)*, 2015. 36, 38, 39, 95, 118, 122, 132

[94] Charles AE Goodhart. Problems of monetary management: the uk experience. 1984. 47

[95] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. In *International Conference on Learning Representations (ICLR)*, 2017. URL `https://openreview.net/forum?id=B184E5qee`. 1

[96] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2013. 2, 95, 118

[97] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 83, 90

[98] Ishaan Gulrajani, Colin Raffel, and Luke Metz. Towards GAN benchmarks which require generalization. In *International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=HkxKH2AcFm`. 69, 90, 91, 92, 196

[99] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On Calibration of Modern Neural Networks. In *International Conference on Machine Learning (ICML)*, 2017. vii, 3, 6, 53, 56, 57, 147, 148, 149, 151, 183

[100] Grzegorz Głuch and Rüdiger Urbanke. Query complexity of adversarial attacks. *arXiv:2010.01039*, 2020. 38

[101] Shai Halevi. Homomorphic encryption. 2017. 171

[102] Shai Halevi and Victor Shoup. Bootstrapping for helib. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2015. 172

[103] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 2011. 128

[104] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016. URL `https://arxiv.org/abs/1510.00149`. 174

[105] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning (ICML)*, 2019. 117

[106] L.K. Hansen and P. Salamon. Neural network ensembles. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990. 60

[107] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv:1711.10677*, 2017. 167

[108] Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *arXiv:1903.08560*, 2019. 96

[109] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision (ICCV)*, 2015. 1, 95, 118

[110] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 65, 82, 102, 116, 130, 161, 173, 195

[111] Dan Hendrycks and Kevin Gimpel. Early methods for detecting adversarial images. 2016. 47

[112] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. 2019. 103

[113] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 95

[114] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 90

[115] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006. 121

[116] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Workshop on Deep Learning and Representation Learning*, 2015. 45, 46

[117] Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. What do compressed deep neural networks forget? *arXiv:1911.05248*, 2019. 108

[118] Harold Hotelling. Canonical correlation analysis (cca). *Journal of Educational Psychology*, 1935. 118

[119] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 82, 161

[120] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 3, 2017. 65, 102, 173

[121] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, University of Massachusetts, Amherst, 2007. 193

[122] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, 2017. 48

[123] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research (JMLR)*, 2016. 174

[124] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv:1803.05961*, 2018. 63

[125] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. 65, 173

[126] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. 2, 67

[127] Jörn-Henrik Jacobsen, Jens Behrmann, Richard S. Zemel, and Matthias Bethge. Excessive invariance causes adversarial vulnerability. In *International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=BkfbpsAcF7`. 117

[128] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. In *British Machine Vision Conference (BMVC)*, 2014. 145

[129] Prateek Jain, Raghu Meka, and Inderjit S Dhillon. Guaranteed rank minimization via singular value projection. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2010. 128

[130] Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. In *European Conference on Computer Vision (ECCV)*, 2018. 44

[131] Byeongmoon Ji, Hyemin Jung, Jihyeun Yoon, Kyungyul Kim, and Younghak Shin. Bin-wise temperature scaling (bts): Improvement in confidence calibration performance through simple scaling techniques. *ICCV 2019 Workshop on Interpreting and Explaining Visual Artificial Intelligence Models*, 2019. 58

[132] Yiding Jiang*, Behnam Neyshabur*, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=SJgIPJBFvH`. 26

[133] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Robust convolutional neural networks under adversarial noise. *arXiv:1511.06306*, 2015. 45

[134] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 2002. 19, 118

[135] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Conference on Security Symposium*, 2018. 168, 170

[136] Sandesh Kamath, Amit Deshpande, and K V Subrahmanyam. Invariance vs robustness of neural networks. *OpenReview*, 2020. URL `https://openreview.net/forum?id=HJxp9kBFDS`. 111

[137] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, 2017. 48

[138] Michael Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 1998. 13

[139] Minje Kim and Paris Smaragdis. Bitwise neural networks. In *ICML Workshop on Resource-Efficient Machine Learning*, 2015. 65, 173

[140] Wonsik Kim, Bhavya Goyal, Kunal Chawla, Jungmin Lee, and Keunjoo Kwon. Attention-based ensemble for deep metric learning. In *European Conference on Computer Vision (ECCV)*, 2018. 60

[141] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014. 195

[142] Patricia Kipnis, Benjamin J. Turk, David A. Wulf, Juan Carlos LaGuardia, Vincent Liu, Matthew M. Churpek, Santiago Romero-Brufau, and Gabriel J. Escobar. Development and validation of an electronic medical record-based alert score for detection of inpatient deterioration outside the ICU. *Journal of Biomedical Informatics*, 2016. 52

[143] V. Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 2001. 27

[144] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 2001. 23, 61

[145] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 82, 100, 115, 161, 190

[146] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2012. 20, 95, 118

[147] Meelis Kull, Miquel Perelló-Nieto, Markus Kängsepp, Telmo de Menezes e Silva Filho, Hao Song, and Peter A. Flach. Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 53, 54, 58

[148] Ananya Kumar, Percy S Liang, and Tengyu Ma. Verified Uncertainty Calibration. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 53, 55

[149] Aviral Kumar, Sunita Sarawagi, and Ujjwal Jain. Trainable Calibration Measures For Neural Networks From Kernel Mean Embeddings. In *International Conference on Machine Learning (ICML)*, 2018. 57, 59, 161, 164

[150] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Ensemble nyström method. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2009. 128, 130

[151] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling techniques for the nystrom method. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009. 130

[152] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. 2016. 39, 41, 42, 132

[153] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *International Conference on Learning Representations (ICLR)*, 2017. 42, 46, 132, 135

[154] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 60

[155] Ken Lang. Newsweeder: Learning to filter netnews. 1995. 161, 192

[156] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. 1998. 2, 3, 100, 167, 190

[157] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *IEEE Symposium on Security and Privacy (SP)*, 2019. 43, 48, 49

[158] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single shot network pruning based on network connectivity. In *International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=B1VZqjAcYX`. 134

[159] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. 2019. 48

[160] Juncheng Li, Frank Schmidt, and Zico Kolter. Adversarial camera stickers: A physical camera-based attack on deep learning systems. In *International Conference on Machine Learning (ICML)*, 2019. 95

[161] Mingchen Li, Mahdi Soltanolkotabi, and Samet Oymak. Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020. 103

[162] Tengyuan Liang and Alexander Rakhlin. Just interpolate: Kernel "ridgeless" regression can generalize. *Annals of Statistics*, 2020. 96

[163] Jae Hyun Lim and Jong Chul Ye. Geometric gan. *arXiv:1705.02894v2*, 2017. 90

[164] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. In *International Conference on Learning Representations (ICLR)*, 2014. URL `https://openreview.net/forum?id=ylE6yojDR5yqX`. 161, 192

[165] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *International Conference on Computer Vision (ICCV)*, 2017. 6, 59, 147, 151, 158

[166] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. Oblivious Neural Network Predictions via MiniONN transformations. In *ACM SIGSAC Conference on Computer and Communications Security*, 2017. 168, 170

[167] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 110

[168] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *International Conference on Computer Vision (ICCV)*, 2015. 83, 192

[169] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 1982. 123

[170] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2015. 1

[171] Chunchuan Lyu, Kaizhu Huang, and Hai-Ning Liang. A unified gradient regularization family for adversarial examples. In *IEEE International Conference on Data Mining (ICDM)*, 2015. 44

[172] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`. 38, 40, 43, 46, 51, 95, 101, 103, 109, 122, 132, 133

[173] Chengzhi Mao, Amogh Gupta, Vikram Nitin, Baishakhi Ray, Shuran Song, Junfeng Yang, and Carl Vondrick. Multitask learning strengthens adversarial robustness. In *European Conference on Computer Vision (ECCV)*, 2020. 109

[174] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. Intel® software guard extensions (intel® SGX) support for dynamic memory management inside an enclave. In *Hardware and Architectural Support for Security and Privacy (HASP*, 2016. 62

[175] Anthony Meehan, Ryan K L Ko, and Geoff Holmes. Deep learning inferences with hybrid homomorphic encryption. *OpenReview*, 2018. URL `https://openreview.net/forum?id=ByCPHrgCW`. 167, 168, 170, 181, 182, 193

[176] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=SyyGPP0TZ`. 2

[177] Carl D Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, 2000. 129

[178] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics*, 1960. 203

[179] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014. 90

[180] R. V. Mises and H. Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung . *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, 1929. 69

[181] Takeru Miyato and Masanori Koyama. cgans with projection discriminator. 2018. 90, 93

[182] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=B1QRgziT-`. 19, 67, 68, 75, 79, 81, 82, 83, 90, 91, 195

[183] Jeet Mohapatra, Ching-Yun Ko, Tsui-Wei, Weng, Sijia Liu, Pin-Yu Chen, and Luca Daniel. Rethinking randomized smoothing for adversarial robustness. *arXiv:2003.01249*, 2020. 49

[184] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy (SP)*, 2017. 168, 170

[185] Omar Montasser, Steve Hanneke, and Nathan Srebro. Vc classes are adversarially robustly learnable, but only improperly. In *Conference on Learning Theory (COLT)*, 2019. 51, 116, 117, 122, 183

[186] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: {A} Simple and Accurate Method to Fool Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016. 36, 40, 41, 95, 118, 132, 135, 136, 137

[187] Vladimir Alekseevich Morozov. *Methods for solving incorrectly posed problems*. Springer Science & Business Media, 2012. 20

[188] Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip H. S. Torr, and Puneet K. Dokania. Calibrating deep neural networks using focal loss. 2020. viii

[189] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 46, 59, 162

[190] Vidya Muthukumar, Kailas Vodrahalli, Vignesh Subramanian, and Anant Sahai. Harmless interpolation of noisy data in regression. *IEEE Journal on Selected Areas in Information Theory*, 2020. 96

[191] Mahdi Pakdaman Naeini, Gregory F Cooper, and Milos Hauskrecht. Obtaining Well Calibrated Probabilities Using Bayesian Binning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2015. 54, 55

[192] Vaishnavh Nagarajan and Zico Kolter. Deterministic PAC-bayesian generalization bounds for deep networks via generalizing noise-resilience. In *International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=Hygn2o0qKX`. 76

[193] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. 190

[194] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015. 68

[195] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory (COLT)*, 2015. 25, 28, 29

[196] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=Skz_WfbCZ`. 5, 25, 30, 31, 32, 67, 88

[197] Alexandru Niculescu-Mizil and Rich Caruana. Predicting Good Probabilities With Supervised Learning. In *International Conference on Machine Learning (ICML)*, 2005. 3, 56

[198] Jeremy Nixon, Michael W. Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. Measuring calibration in deep learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 55

[199] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations (ICLR)*, 2018. 76

[200] P. E. Nüesch. Order restricted statistical inference. *Journal of Applied Econometrics*, 1991. 58

[201] Ozan Oktay, Jay Nanavati, Anton Schwaighofer, David Carter, Melissa Bristow, Ryutaro Tanno, Rajesh Jena, Gill Barnett, David Noble, Yvonne Rimmer, Ben Glocker, Kenton O'Hara, Christopher Bishop, Javier Alvarez-Valle, and Aditya Nori. Evaluation of deep learning to augment image-guided radiotherapy for head and neck and prostate cancers. *JAMA Network Open*, 2020. 52

[202] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua V Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 57, 60

[203] Edouard Oyallon. Building a regular decision boundary with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 132

[204] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *arXiv:1511.04508*, 2015. 95

[205] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv:1605.07277*, 2016. 36, 95, 118, 132

[206] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (SP)*, 2016. 46

[207] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ACM Asia Conference on Computer and Communications Security*, 2017. 95

[208] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 192

[209] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. In *International Conference on Learning Representations (ICLR)*, 2017. 21, 152

[210] Hieu Pham, Qizhe Xie, Zihang Dai, and Quoc V. Le. Meta pseudo labels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1

[211] John Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Advances in Large Margin Classifiers*, 1999. 57

[212] Adrian E. Raftery, Tilmann Gneiting, Fadoua Balabdaoui, and Michael Polakowski. Using bayesian model averaging to calibrate forecast ensembles. *Monthly Weather Review*, 2005. 60

[213] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018. 48

[214] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C. Duchi, and Percy Liang. Adversarial training can hurt generalization. *arXiv:1906.06032*, 2019. 104

[215] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *Worshop Track in the International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SkBYYyZRZ`. 196

[216] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019. 1

[217] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015. 95, 118

[218] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *ACM Asia Conference on Computer and Communications Security*, 2017. 168, 170

[219] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 1978. 64

[220] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2015. 1

[221] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. 2018. 44

[222] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Adversarially robust training through structured gradient regularization. *arXiv:1805.08736*, 2019. 44

[223] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. DeepSecure: Scalable Provably-Secure Deep Learning. In *ACM International Symposium on Design Automation Conference (DAC)* , 2017. 170

[224] Mark Rudelson and Roman Vershynin. Sampling from large matrices. *Journal of the ACM*, 2007. 5, 71

[225] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. 1

[226] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 67

[227] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 90, 91

[228] Swami Sankaranarayanan, Arpit Jain, Rama Chellappa, and Ser Nam Lim. Regularizing deep networks using efficient layerwise adversarial training. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017. 45

[229] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 67

[230] Amartya Sanyal, Puneet Dokania, Varun Kanade, and Philip H.S. Torr. Robustness via deep low-rank representations. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018. viii

[231] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. TAPAS: Tricks to Accelerated (encrypted) Prediction As a Service. In *International Conference on Machine Learning (ICML)*, 2018. viii

[232] Amartya Sanyal, Philip H. Torr, and Puneet K. Dokania. Stable rank normalization for improved generalization in neural networks and {gan}s. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=H1enKkrFDB`. viii, 134, 146

[233] Amartya Sanyal, Puneet K Dokania, Varun Kanade, and Philip H. S. Torr. How benign is benign overfitting? In *International Conference on Learning Representations (ICLR)*, 2021. viii

[234] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 50, 109, 113, 122, 183

[235] Lea Schönherr, Katharina Kohls, Steffen Zeiler, Thorsten Holz, and Dorothea Kolossa. Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding. In *Network and Distributed System Security Symposium*, 2018. 95

[236] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Annual Meeting of the Association for Computational Linguistics*, 2015. 2

[237] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2014. URL `https://openreview.net/forum?id=Hq5MgBFOP62-X`. 118

[238] Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020. 109, 111, 113

[239] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 82, 102

[240] Samarth Sinha, Homanga Bharadhwaj, Anirudh Goyal, Hugo Larochelle, Animesh Garg, and Florian Shkurti. Diversity inducing information bottleneck in model ensembles. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020. 60

[241] Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Designs, codes and cryptography*, 2014. 65

[242] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013. 161, 192

[243] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 2014. 2

[244] Asa Cooper Stickland and Iain Murray. Diverse ensembles improve calibration. In *ICML Workshop on Uncertainty and Robustness in Deep Learning*, 2020. 60

[245] Beata Strack, Jonathan P. DeShazo, Chris Gennings, Juan L. Olmo, Sebastian Ventura, Krzysztof J. Cios, and John N. Clore. Impact of HbA1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records. *BioMed Research International*, 2014. 193

[246] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press Wellesley, MA, 1993. 185

[247] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. In *Annual Meeting of the Association for Computational Linguistics*, 2019. 20

[248] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2013. vii, 3, 36, 40, 44, 95, 118, 132

[249] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2, 45, 46, 183

[250] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and Weinan E. Convolutional neural networks with low-rank regularization. In *International Conference on Learning Representations (ICLR)*, 2016. 161, 192

[251] TAPAS. TAPAS – tricks for accelerating (encrypted) prediction as a service. https://github.com/amartya18x/tapas, 2018. 179

[252] Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On Mixup Training: Improved Calibration and Predictive Uncertainty for Deep Neural Networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 59

[253] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2017. 48

[254] Nikhil Kumar Tomar, Debesh Jha, Michael A. Riegler, Håvard D. Johansen, Dag Johansen, Jens Rittscher, Pål Halvorsen, and Sharib Ali. Fanet: A feedback attention network for improved biomedical image segmentation. *arXiv:2103.17235*, 2021. 1

[255] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. 2020. 47, 95

[256] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=rkZvSe-RZ. 95, 133

[257] Dustin Tran, Rajesh Ranganath, and David M. Blei. Hierarchical implicit models and likelihood-free variational inference. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 90

[258] TreeLSTM. TreeLSTM. https://github.com/ttpro1995/TreeLSTMSentiment, 2015. Accessed: 2019-05-22. 192

[259] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=SyxAb30cY7. 50, 104, 109, 115, 116, 122, 191

[260] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 44

[261] Juozas Vaicenavicius, David Widmann, Carl Andersson, Fredrik Lindsten, Jacob Roll, and Thomas B Schön. Evaluating model calibration in classification. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019. 53

[262] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 1984. 9

[263] Bart van Merriënboer, Amartya Sanyal, Hugo Larochelle, and Yoshua Bengio. Multiscale sequence modeling with a learned dictionary. In *ICML Workshop on Machine Learning in Speech and Language Processing*, 2017. 2

[264] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Conference on Neural Information Processing Systems (NeurIPS)*, 1992. 13

[265] Vladimir Vapnik. *Statistical learning theory. 1998*. Wiley, New York, 1998. 13

[266] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 2, 95, 118

[267] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning (ICML)*, 2013. 181, 182

[268] Chenguang Wang, Mu Li, and Alexander J. Smola. Language models with transformers. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2019. 1

[269] Colin Wei and Tengyu Ma. Data-dependent sample complexity of deep neural networks via lipschitz augmentation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 68, 76, 88

[270] F. Wenzel, Kevin Roth, Bastiaan S. Veeling, J. Swiatkowski, L. Tran, S. Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and S. Nowozin. How good is the bayes posterior in deep neural networks really? In *International Conference on Machine Learning (ICML)*, 2020. 60

[271] David Widmann, Fredrik Lindsten, and Dave Zachariah. Calibration tests in multi-class classification: A unifying framework. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 53

[272] Helmut Wielandt. An extremum property of sums of eigenvalues. In *Proceedings of the American Mathematical Society*, 1955. 203

[273] Christopher K I Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2001. 128

[274] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. 2020. 60

[275] Lior Wolf, Tal Hassner, and Yaniv Taigman. Effective unconstrained face recognition by combining multiple descriptors and learned background statistics. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011. 193

[276] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, 2018. 48

[277] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=BJx040EFvH`. 103

[278] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*, 2016. 1

[279] Greg Yang, Tony Duan, Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. Randomized smoothing of all shapes and sizes. In *International Conference on Machine Learning (ICML)*, 2020. 49

[280] Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Ruslan Salakhutdinov, and Kamalika Chaudhuri. Adversarial robustness through local lipschitzness. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020. 109

[281] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=HkwZSG-CZ`. 1

[282] Zhuolin Yang, Bo Li, Pin-Yu Chen, and Dawn Song. Characterizing audio adversarial examples using temporal dependency. In *International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=r1g4E3C9t7`. 47

[283] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Annual Symposium on Foundations of Computer Science (FOCS*, 1986. 63

[284] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 2007. 18

[285] Dong Yin, Ramchandran Kannan, and Peter Bartlett. Rademacher complexity for adversarially robust generalization. In *International Conference on Machine Learning (ICML)*, 2019. 109

[286] Xuwang Yin, Soheil Kolouri, and Gustavo K. Rohde. Adversarial example detection and classification with asymmetrical adversarial training. *arXiv:1905.11475*, 2019. 47

[287] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv:1705.10941, 2017.*, 2017. 20

[288] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *International Conference on Machine Learning (ICML)*, 2001. 58

[289] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016. 82, 95, 161

[290] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2014. 118

[291] Chiyuan Zhang and Vitaly Feldman. What neural networks memorise and why: Discovering the long tail via influence estimation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020. 104, 105, 106, 107

[292] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. 2016. vii, 3, 16, 17, 24, 28, 85, 95, 110, 183

[293] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International Conference on Machine Learning (ICML)*, 2018. 90

[294] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning (ICML)*, 2019. 40, 43, 46, 50, 95, 103, 109, 122

[295] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=r1Ddp1-Rb`. 59

[296] Shuxin Zheng, Qi Meng, Huishuai Zhang, Wei Chen, Nenghai Yu, and Tie-Yan Liu. Capacity control of relu neural networks by basis-path norm. 2018. 68

[297] Leon Wenliang Zhong and James T. Kwok. Accurate probability calibration for multiple classifiers. 2013. 60

[298] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*, 2017. 174