

Simplicity bias and generalization in deep learning



Guillermo Jorge Valle Pérez
Kellogg College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Trinity 2020

Acknowledgements

I would like to thank my supervisor Ard Louis for encouraging me to explore the interesting, rather than the easy, paths; and Chris Mingard and Joar Skalse for the countless hours discussing ideas on the chalkboard late in the physics department; as well as my other amazing colleagues including Chico Camargo, Kamal Dingle, Yoav Rubinstein, Shuofeng Zhang, and David Martinez.

I would also like to thank the whole cohort of Systems Biology 2016 for being the most warm and welcoming group of people I've ever met. 2016 was an amazing year thanks to you, and thanks to the people in the Woodstock Road house. I would like to thank Paco Criado and David Martinez for being so awesome and exploring *every interesting idea wherever it took us*. Thanks as well to Laura Gonzalez, Mateo Galdeano, Paula Párraga, and Alex Garcia for all the fun boardgame nights. Thanks to Joaquin Pons, Adria Garriga, and Paco, for best memes.

Thanks to Avi Roy for being so awesome and inspiring, and knowing and introducing me to all the coolest people, including Anders Sandberg, Eric Drexler, Stuart Armstrong, Diana Sofronieva, Cristina Castro, as well as the rest of the worldwide transhumanist community! Special thanks to Cristina Castro and Alex Ivanov for all the fun and stimulating conversations and walks.

I'd like to thank the whole OxAI community, including Rodrigo Mendoza, Shin Huang, Sewook Oh, Daniel Lozano, Hailey Trier, Kai Zhi Teh, Ruben Nathaniel Drayton, Ralph Abboud, Tim Seabrook, and Rian Hughes, for an incredible journey in engaging the AI community in Oxford and beyond, and working on awesome projects. Thanks to Shin for being "partner in crime" in so many adventures.

Thanks to the whole NeosVR community (and to Alex Cheema for introducing me to it), specially Frooxius and Karel who have started something so amazing, and to Nexulan, Aegis, Coffee, Outsider, Medra, Sirkitree, Ukilop, Lucas, Earthmark, Starkido, Hamish, Joris, Neosmotic, Zyzyl, jeana, 3x1t, goodvibes, Staccato, alex derpy avali, and many more, for all the many fun moments in this magically weird place called Neos.

Thanks as well to the G club, Nikolai Drouzine, and Bruno Simonetta for being dank and awesome friends (uwu).

Last, but definitely not least, thanks to my family, my parents Marusela and Carlos, and my sister Alejandra. I can't thank you enough for all your love and support throughout not just the DPhil, but life in general.

Abstract

Deep learning is revolutionizing many fields of application of machine learning. On the other hand, theory is struggling to keep pace with the rate of practical success. In particular, deep neural networks often work best when they are overparametrized – an observation that can not be explained by standard worst-case learning theory. The origin of an inductive bias that would explain the generalization performance of these models has become a central theoretical question in the field. In this work we propose and develop a new theory to explain the generalization performance of deep learning models. Supported by several empirical and theoretical findings, we claim that the main source of inductive bias in deep learning models is the parameter-function map, which refers to the map mapping parameters of the neural network to the function it implements. We study the bias in this map, showing that simple functions have many more parameters producing them, than complex functions. We develop this idea into a quantitative theory of generalization in deep learning, based on PAC-Bayes theory and the mean-field approximation of wide neural networks, as well as the observation that standard optimizers used for DNNs sample functions in an approximately Bayesian fashion. We empirically validate this theory and find that it predicts the observed generalization error significantly better than competing theories.

Contents

1	Introduction	1
1.0.1	Early history of artificial neural networks	2
1.0.2	Learning theory and ANNs	4
1.0.3	Machine learning and algorithmic information theory	7
1.0.4	Towards a new theory of generalisation for ANNs	8
2	Simplicity bias in the parameter-function map	12
2.1	Deep neural networks and the parameter-function map	13
2.2	Algorithmic information theory and simplicity-bias in the parameter-function map	14
2.3	Empirical study of simplicity bias in a DNN implementing Boolean functions .	17
2.4	Effects of target function complexity on learning	20
2.4.1	Lempel-Ziv versus Entropy	23
2.5	Simplicity bias in larger systems	24
3	The perceptron parameter-function map	28
3.1	Summary of key results	29
3.2	Definitions, Terminology, and Notation	30
3.3	Intrinsic bias in a perceptron’s parameter-function map	31
3.3.1	Entropy bias in a simple perceptron with $b = 0$ (no threshold bias term)	32
3.3.2	Simplicity bias of the $b = 0$ perceptron	33
3.3.3	Bias within \mathbb{F}_t	35
3.3.4	Effect of b (the threshold bias term) on $P(t)$	37
3.4	Entropy bias in Multi-layer Neural Networks	38

3.4.1	Expressivity conditions for DNNs	39
3.4.2	How multiple layers affect the bias	39
3.5	Bias towards low entropy in realistic datasets and architectures	42
3.6	Effect of the bias term on the perceptron on centered data	44
3.7	Effect of bias on learning	45
4	Generalization theory of deep learning	49
4.1	Desiderata for predictive theories of generalization error	52
4.2	Classifying generalization bounds for deep learning	54
4.2.1	Notation for the supervised learning problem	55
4.2.2	General PAC learning framework	56
4.2.3	A formal classification of generalization bounds	57
4.2.3.1	According to assumptions on \mathcal{D}	57
4.2.3.2	According to assumptions on \mathcal{A}	58
4.2.3.3	According to dependence of capacity on S	60
4.2.3.4	Non-uniform bounds and algorithm-dependent bounds	61
4.2.3.5	Overview of bounds	63
4.3	Comparing existing bounds against desiderata	64
4.3.1	Algorithm-independent bounds	64
4.3.1.1	Data-independent uniform convergence bounds: VC dimension	64
4.3.1.2	Data-dependent uniform convergence bounds: Rademacher complexity	66
4.3.2	Algorithm-dependent bounds	68
4.3.2.1	Algorithm-dependent bounds based on non-uniform convergence	68
4.3.2.2	Other PAC-Bayes bounds	81
4.3.3	Other algorithm dependent bounds	81
4.3.3.1	Stability-based bounds	81
4.3.4	Other bounds and generalization measures	85
4.3.5	Generalization error predictions for specific data distributions	86
4.3.6	Optimality of data-dependent bounds	87

4.4	Marginal-likelihood PAC-Bayesian generalization error bound	88
4.5	Bayesian optimality of PAC-Bayesian bound	90
5	PAC-Bayes experiments	94
5.0.1	Error versus label corruption (Desideratum D.1)	95
5.0.2	Error versus training set size (Desideratum D.2)	96
5.0.3	Error versus architecture (Desideratum D.3)	102
5.1	Discussion of the results	104
6	Does SGD approximate Bayesian sampling?	107
6.1	Main results summary	110
6.2	Preliminaries	111
6.2.1	The Bayesian prior probability, $P(f)$	111
6.2.2	The Bayesian posterior probability, $P_B(f S)$	112
6.2.3	The optimiser probability, $P_{\text{OPT}}(f S)$	113
6.3	Methodology	113
6.3.1	Definition of functions	113
6.3.2	Calculating $P_{\text{OPT}}(f S)$	114
6.3.3	Calculating $P_B(f S)$ with Gaussian Processes	115
6.3.4	Comparing $P_{\text{OPT}}(f S)$ to $P_B(f S)$	116
6.3.5	Calculating $P_B(f S)$ for functions with generalisation error from 0% to 100%	116
6.3.6	CSR complexity	117
6.4	Empirical results for $P_B(f S)$ v.s. $P_{\text{OPT}}(f S)$ for different architectures and datasets	117
6.4.1	Comparing $P_B(f S)$ to $P_{\text{OPT}}(f S)$ for FCN on MNIST	117
6.4.2	Comparing $P_B(f S)$ to $P_{\text{Adam}}(f S)$ for CNNs on Fashion-MNIST	122
6.4.3	Comparing $P_B(f S)$ and $P_{\text{SGD}}(f S)$ to Neural Tangent Kernel results	123
6.4.4	Comparing $P_B(f S)$ to $P_{\text{Adam}}(f S)$ for LSTM on IMDb sentiment analysis	125
6.4.5	Comparing $P_B(f S)$ to $P_{\text{Adam}}(f S)$ for FCN on Ionosphere dataset	126
6.4.6	Effects of training set size	126

6.4.7	Results for other test sets	126
6.5	The effect of hyperparameter changes and optimisers on $P_B(f S)$ and $P_{\text{OPT}}(f S)$	127
6.5.1	Changing batch size and learning rate	127
6.5.2	Changing optimisers	128
6.6	Heuristic arguments for the correlation between $\mathbf{P}_B(\mathbf{f} \mathbf{S})$ and $\mathbf{P}_{\text{SGD}}(\mathbf{f} \mathbf{S})$	128
6.7	Related work on inductive bias on neural networks	132
6.7.1	The link between inductive bias and generalisation	132
6.7.2	Related work on implicit bias in optimiser-trained networks	133
6.7.3	Related work on implicit bias in random neural networks	136
6.7.4	Related work comparing optimiser-trained and Bayesian neural networks	138
6.7.5	Related work on complexity of data, simplicity bias and generalisation	139
7	Discussion	141
7.1	Simplicity bias in the parameter-function map	142
7.2	Generalization theory of DNNs and marginal-likelihood bound	144
7.2.1	Why the marginal-likelihood PAC-Bayes bound works well	145
7.3	SGD-trained networks are approximately Bayesian	147
7.4	Conclusion	148
Appendices		150
Appendix A	Experiment details	151
A.1	Experiment details for Chapter 2	151
A.1.1	Finite-size effects for sampling probability	152
A.2	Experimental details for Chapter 5	152
A.2.1	Training, dataset, and architecture details	153
A.3	Further detail for experiments in Chapter 6	155
A.3.1	Data sets	156
A.3.2	Architectures	156
A.3.3	Methodology in detail	157
A.3.3.1	Using an optimiser to calculate $\mathbf{P}_{\text{OPT}}(\mathbf{f} \mathbf{S})$	157

A.3.3.2	Bayesian sampling for $\mathbf{P}_B(\mathbf{f} \mathbf{S})$	158
A.3.3.3	Calculating $P_B(f S)$ for functions with a wider range of ϵ_G	159
A.3.3.4	Further notes on methods	161
Appendix B Mean field theory of DNNs		162
B.1	Bayesian framework	162
B.2	Neural network Gaussian processes	164
B.2.1	Computing the kernel of NNGPs	165
B.3	Computation of posterior and marginal likelihood	166
B.3.1	NNGP with 0-1 likelihood and EP approximation	166
B.3.2	NNGP with Gaussian likelihood	168
B.4	Empirical results concerning the GP approximations	169
B.4.1	Comparing approximate GP marginal likelihoods with direct sampling for MNIST	172
Appendix C Proofs		174
C.1	Proof of uniformity	174
C.2	$P(t = 0)$ for perceptron with infinitesimal b	177
C.3	Bounding Boolean function complexity, K_{Bool} , with t	178
C.4	Theorems associated with entropy increase for DNNs	181
C.5	Proof of Theorem 4.4.1	189
C.6	Proof of Theorem 4.5.1	190
C.7	Derivation of KL divergence inequality	191
Appendix D Further results for the simplicity bias in the simple Boolean sys- tem		192
D.1	Other complexity measures	192
D.1.1	Complexity measures	193
D.1.2	Correlation between complexities	195
D.1.3	Probability-complexity plots	195
D.2	Effect of number of layers on simplicity bias	195

D.3	Effect of target function complexity on learning for different complexities	200
D.4	Probability-complexity plots for other parameter distributions	201
D.5	Number of iterations to learn a function vs depth and width	203
D.6	Bias and the curse of dimensionality	204
Appendix E	Further results for the perceptron	206
E.0.1	Zipf's law in a perceptron with $b = 0$	206
E.1	Bias disappears in the chaotic regime	209
E.2	Further results on the distribution within \mathbb{F}_t	212
E.2.1	Empirical results	212
E.2.2	Substructure within $\{\mathbf{0}, \mathbf{1}\}^n$	215
E.3	Towards understanding the simplicity bias observed within \mathbb{F}_t	217
Appendix F	Further results on PAC-Bayes learning curves	223
F.1	Learning curves versus dataset for all resnets	224
F.2	Learning curves versus dataset for all densenets	225
F.3	Error versus bound for batch size 32 for more datasets	225
F.4	Learning curves versus dataset for all architectures	229
F.5	Learning curves for batch size 256	231
F.6	Error versus bound for batch size 256	238
F.7	Error versus bound for batch size 32, including CNN and FCN	242
F.8	Error versus bound for batch size 256, including CNN	247
Appendix G	Further results on comparing SGD and Bayesian inference	253
G.1	Further results comparing $P_{\text{OPT}}(f S)$ to $P_{\text{B}}(f S)$	254
G.1.1	Changing optimisers	263
G.1.2	Effects of training set size	264
G.2	Bayesian inference by direct sampling on Boolean system.	264
G.3	Critical Sample Ratio	273
G.4	Notes on the distribution of MNIST data	274

Appendix H The error-complexity plane	277
H.1 The error-complexity landscape	277
H.1.1 Error-entropy plane	278
H.1.2 Error-Kolmogorov complexity plane	279
H.2 What functions does the network find?	280
H.3 Bounds and number of functions in the error-entropy plane	284
H.4 Bounds for the error-Kolmogorov complexity plane	285
H.5 Probability of choosing a function of given error and entropy	286
H.6 Lower bound on the complexity of functions compatible with the training set .	288
H.7 Error-complexity histograms	290
Bibliography	293

Chapter 1

Introduction

In the last decade, machine learning and artificial intelligence (AI) have experienced a surge in research activity and investment, largely due to the subfield of deep learning. Deep learning and AI have a long and interdisciplinary history, interacting with fields from neuroscience to physics. In fact, it is argued that the current success in deep learning is mostly due to a recent increase in the amount of data and compute capabilities, while many of the ideas and questions are much older. In 1995, the influential statistician Leo Brieman wrote an article titled “Reflections After Refereeing Papers for NIPS” [Breiman, 1995] in which he poses some of the main open questions of the day in machine learning theory:

- Why don’t heavily parameterized neural networks overfit the data?
- What is the effective number of parameters?
- Why doesn’t backpropagation head for a poor local minima?
- When should one stop the backpropagation and use the current parameters?

Even though these questions were written 25 years ago, they are still open and are some of the main topics studied by recent papers in the theory of deep neural networks. In that article, Brieman also expresses his opinions on the value of theory, and how achieving understanding of new phenomena, like machine learning systems, relies on much more than mathematically rigorous theorems. He proposes inquiry techniques such as: “mathematical heuristics, simplified

analogies (like the Ising Model), simulations, comparisons of methodologies, devising new tools, theorems where useful (rare!), and shunning panaceas” [Breiman, 1995].

Recently, in a commentary in Nature Physics, Lenka Zdeborová [Zdeborová, 2020] builds on this idea to propose that physicists should play a key role in the quest to understand deep learning. Physics typically employs a diversity of inquiry techniques as Breiman suggests, while maintaining a high standard of scientific rigour. The study of machine learning systems is particularly connected to the study of complex systems and emergence in physics. In both of these areas, it is often the case that the microscopic rules of a system are known precisely, but the macroscopic behaviour is still surprising, and one would like to find a simple explanation of it. A mathematical theorem is only one approach at tackling this problem. In physics, a more common approach relies on the scientific method: performing carefully designed experiments to rule out possible hypotheses. This approach may not give quite the same level of confidence as a mathematical theorem, but it can still provide insight into observed phenomena, produce non-trivial and correct predictions, and may also inspire new theorems.

1.0.1 Early history of artificial neural networks

The history of artificial neural networks (ANNs) can be traced to the classic paper McCulloch and Pitts [1943] in which they introduced a model of networks of neurons based on simplifying assumptions of real physiological neurons. These neurons had binary 0 or 1 activations, and discrete “weights” (corresponding to number of “synapses”). McCulloch and Pitts didn’t propose an explicit training mechanism for these networks. The first learning algorithm for neural networks was proposed by Hebb [1949], based on the idea that “neurons that fire together wire together”. Interestingly a year earlier, Alan Turing, considered one of the founding fathers of AI, had published a report [Turing, 1948] in which he introduced a learning machine similar to ANNs, and speculated about how these could learn. Frank Rosenblatt proposed in 1959 a model which he called the perceptron [Rosenblatt, 1958], consisting of a network of neurons like that of McCulloch and Pitts, and a new training algorithm, inspired by the work of Hebb, and other literature on “brain models” [Ashby, 1952, Hayek, 1952, Uttley, 1956]. He considered neural networks with hidden layers, but only the last layer was trained, and the parameters corresponding to biases in the neurons only, rather than weights in the synapses. His model

was simple enough that it allowed theoretical analysis of its training and generalization learning curves [Rosenblatt, 1961, Joseph, 1960, Lumb, 1959]. Rosenblatt’s paper was very influential and inspired several works on extensions to the perceptron.

It was quickly recognized that multiple layers allowed for more expressivity and better generalization [Rosenblatt, 1961, Bryan, 1963]. However, training parameters across layers remained a challenge [Rosenblatt, 1961]. Some early attempts at training several layers include the Gamba networks [Gamba, 1961, Palmieri and Sanna, 1960], and networks of Adaline neurons [Widrow and Hoff, 1960, Widrow, 1962]. The Adaline neurons [Widrow and Hoff, 1960] are interesting in that they were inspired by earlier work on switching networks by Shannon [1938] and on adaptive neuron models by Von Neumann [1956] and Mattson [1959], Mattson, they are described in a way which is closer to the modern interpretation of the “perceptron”, and they were trained with a version of gradient descent (in the case of no hidden layers). In 1965, Ivakhnenko et al. published a training algorithm for adjusting all the weights of multi-layer neural networks, which they later used to successfully train deep neural networks (DNNs) of up to 8 layers [Ivakhnenko and Lapa, 1965]. This work appears to have been unknown to Minsky and Papert when they published their influential book on perceptrons 4 years later [Marvin and Seymour, 1969]. In this book they analyzed the theoretical limitations of a class of perceptrons in which the features feeding to the trained linear classifier were assumed to be local. They commented on other classes of perceptrons including the Gamba networks, and more general multilayer perceptrons, speculating that these extensions are probably not promising, although they could not yet offer proof of this due to the difficulty in analysing these systems. This book is claimed to have caused a huge drop in funding for work on neural networks, and shifted most focus on AI research to systems based on logical rules, deduction, and explicit heuristics [Olazaran, 1996, Cohen]. The high expectation placed on this approach, combined with the comparatively poor results, led to the first “AI winter” during the decade of the 70s [Howe, 2007]. In the 80s, there was renewed interest in the rule-based approach to AI with the development of expert systems, but these systems also proved too brittle, resulting in a second AI winter at the end of the 80s [Stuart et al., 2003].

Despite the sharp decrease in funding for research into neural networks and related “connectionist” approaches during these two decades, some important advances were made that laid

the foundations for later theoretical and applied developments. On the theoretical side, William Little proposed in 1974 an analogy between the Ising model of spin glasses and neural networks [Little, 1974]. We also saw the development of neural networks with emergent collective behaviour that allowed to recover a previously seen pattern, from a subpart [Amari, 1977, Kohonen, 1972]. This type of “associative” or “content-addressable” memory inspired the work of Hopfield [1982], which popularized the Ising-like model of Little, and in turn inspired an increasing number of physicists to work on the theory of neural networks [Sompolinsky, 1988]. In 1989, expressivity theorems were developed [Hornik et al., 1989], based on earlier work by Kolmogorov [1957], which generalized the older results on approximating Boolean functions to real-valued functions. On the more applied side, the 70s and 80s saw the development of backpropagation [Linnainmaa, 1970, Werbos, 1974, 1982, Parker, 1985, Rumelhart et al., 1985], and convolutions [Fukushima, 1979, LeCun et al., 1989]. These developments ushered in a big resurgence of work on neural networks during the 90s, including recurrent neural networks (RNNs) and increasingly performant architectures and algorithms, that ultimately led to the current ‘AI spring’ based on deep learning, starting in the early 2010s. We refer the reader to the articles Schmidhuber [2015], LeCun et al. [2015] for a good review of these developments, focused on practical developments.

1.0.2 Learning theory and ANNs

The theory of neural networks is tightly linked to the theory of machine learning. The latter traditionally encompasses three main subareas: supervised learning, unsupervised learning, and reinforcement learning. We will focus on supervised learning because it is arguably the most studied and applied of the three areas, and holds many of the foundational questions which are relevant throughout all of machine learning. As we mentioned earlier, work on predicting the learning properties of neural networks goes back at least to the original paper on perceptrons [Rosenblatt, 1958]. Rosenblatt formalized “learning” as equivalent to “generalization”, which he defines as being able to predict correct responses to inputs not seen during training. He distinguished that from the ability to correctly predict responses to inputs in the training set (for which it was given the correct answer, or at least a reinforcing signal to learn the correct answer). This separation of the learning problem into fitting the training data (an optimization

problem), and generalizing to new data (an statistical problem) is still used as the basis of most analyses of supervised learning – see the modern formalization we present in Section 4.2.1.

Since the 80s, theory of supervised learning neural networks began developing in two main branches. On the one hand, there was the work following the approach from statistical physics [Hopfield, 1982, Sompolinsky, 1988, Tishby, 1995, Smieja, 1989, Wallace, 1987], which was often less rigorous, and based on extensive experiments, as well as focusing on “average-case” analysis. On the other hand, the development of learning theory as a branch of statistics by Vapnik and Chervonenkis [Vapnik, 1968, Vapnik and Chervonenkis, 1974, Vapnik, 1995], and of theoretical computer science by Leslie Valiant [Valiant, 1984], took a very rigorous approach, mostly based on theorems, and “worst-case” analysis with minimal assumptions. Vapnik and Chervonenkis’ theory gave rise to the important concept of VC dimension, as measure of the ‘richness’ of a family of classifiers, while Valiant introduced probably approximately correct (PAC) learning, which formalized the concept of ‘generalization’, using frequentist statistics ideas to minimize the number of assumptions. Already in a workshop on the theory of generalization in NIPS 1992, it was recognized that these two communities, as well as others, should find more effective ways to communicate and exchange ideas [Wolpert, 1995]. Although the situation has improved today, we see from the article by Zdeborova with which we opened the thesis [Zdeborová, 2020], that better collaboration between the different disciplines is still a live issue today.

In 1989, Levin, Tishby and Solla introduced a probabilistic view of learning based on a formal analogy between Bayesian inference and statistical physics Tishby et al. [1989], Levin et al. [1990]. This framework led David MacKay in 1992 to propose a way to do model selection on neural networks by selecting the solutions where the error surface was flatter, which he associated with larger Bayesian evidence MacKay [1992a]. MacKay found that “empirically, the correlation between the evidence and generalisation is often good. But a theoretical connection between the two is not yet established”. Two years later, Hochreiter and Schmidhuber developed algorithms to find flat minima of ANNs, and justified their performance with the minimum description length (MDL) principle, and the observation that real-world problems tended to be “simple” and are far from uniformly distributed [Hochreiter and Schmidhuber, 1994, 1995, 1997a]. However, a mathematically rigorous result that could explain this correlation in some cases was first proposed by John Shawe-Taylor in his PAC analysis of a Bayesian estimator

Shawe-Taylor and Williamson [1997] (based on an extension to the structural risk minimization framework [Shawe-Taylor et al., 1998] of Vapnik and Chervonenkis). A similar but more general result was introduced in 1998 by McAllister with his PAC-Bayes theorems [McAllester, 1998]. As we will see in Chapter 4, both flatness of minima, and PAC-Bayes play a big role in recent literature on deep learning theory Keskar et al. [2016], Neyshabur et al. [2017], Bartlett et al. [2017], Dziugaite and Roy [2017], Zhou et al. [2018].

The work of MacKay on Bayesian priors [MacKay, 1992a] also inspired another line of work starting with Neal [1994]. In this paper, he switched focus from the prior over parameters to the prior *over functions*, and he showed that for of a one-hidden-layer ANN, the later approached a Gaussian process when the number of hidden neurons grew to infinity. The result of Neal was recently extended to DNNs of many layers [Lee et al., 2017], and forms the foundation of the mean field theory (MFT) of DNNs, which gives analytical results on the infinite-width limit of the prior over functions for DNNs. The MFT of DNNs is reviewed in Chapter B. This theory has inspired a lot of recent work on deep learning theory [Lee et al., 2017, Schoenholz et al., 2017, Matthews et al., 2018, Garriga-Alonso et al., 2019, Novak et al., 2019a, Yang, 2019a].

Neal and MacKay’s Bayesian approach was a more principled way to avoid ‘overfitting’ and helped popularize the use of Gaussian processes (GPs) in the late 90s and early 2000s [Rasmussen, 2004]. During this same time, support vector machines (SVMs) were developed by Vapnik and others [Boser et al., 1992, Drucker et al., 1997] as a way to perform classification and regression, which came with rigorous generalization guarantees based on VC dimension theory Vapnik [1995]. In part because of their theoretical foundations, GPs and SVMs became increasingly popular in the 2000s, under the common name of “kernel machines” Rasmussen [2004].

This changed in the early 2010s, when a series of ANNs with many layers started winning several international machine learning competitions, and producing breakthroughs in classic AI tasks like computer vision [Cireşan et al., 2011, Krizhevsky et al., 2012], playing games [Mnih et al., 2013], and speech recognition [Hannun et al., 2014]. Because these ANNs have many layers, they are called ‘deep’, and their use is called ‘deep learning’. The success of deep learning has kept growing and now finds applications in a great many academic and applied disciplines [Pierson and Gashler, 2017, Miotto et al., 2018, Yannakakis and Togelius, 2018,

Guest et al., 2018, Foster, 2019, Robila and Robila, 2019].

This surge in interest and research revived several theoretical questions including those posed by Brieman in 1995. One of the most influential articles in this regard was Zhang et al. [2017a] which showed that modern neural networks with millions of parameters have a huge capacity to fit even random labellings of millions of images, and that this implied that existing theory (mostly based on VC dimension) was not able to explain their generalization. In particular, VC dimension theory did not take into account that a model could fail to generalize for some data sets while generalizing well for others, which is precisely what Zhang et al. found about DNNs. The varied approaches that have been tried to solve this problem are presented in a unified manner in Section 4.2.

1.0.3 Machine learning and algorithmic information theory

Another line of work that will be relevant for this thesis applies ideas from algorithmic information theory (AIT) to machine learning. AIT studies a notion of information content for an object defined as the length of the shortest program producing a representation of that object. This quantity is called the Kolmogorov complexity of the object, after Andrei Kolmogorov who proposed it in 1965 and developed a theory analogous to Shannon’s information theory based on it [Kolmogorov, 1965]. In 1964, Solomonoff introduced the related concept of universal a priori probability, which argues that a good prior to use in inductive inference is the prior that assigns probabilities to hypotheses based on their probability that they are produced by a Turing machine which is fed random inputs [Solomonoff, 1964]. In the 1970s, Levin made significant contributions to the field, including the coding theorem [Zvonkin and Levin, 1970, Ming and Vitányi, 2014] which shows a tight connection between the universal prior and Kolmogorov complexity. With regards to applying AIT to machine learning, Hutter has made many contributions [Hutter, 2004], including proofs that an ideal algorithm which is perfectly biased towards simple functions will generalize when the true function is simple [Lattimore and Hutter, 2013]. This result is the AIT version of a similar result based on PAC learning by Blumer et al. [1987], where it was shown that algorithms perfectly biased towards functions with short codes (for any fixed but arbitrary code on the set of possible functions) will generalize

in the PAC sense¹ when the target function also has a short code. Recently, a version of the Levin’s coding theorem for computable functions [Gács, 1988, Dingle et al., 2018a] was applied to a variety of real-world input-output maps and it was found that many systems show a bias towards simple outputs in a way which follows the coding theorem’s predictions [Dingle et al., 2018a].

1.0.4 Towards a new theory of generalisation for ANNs

In this thesis, I will explore the question of why deep neural networks generalize. A lot of recent work has explored this question, but the question is still considered largely open [Kawaguchi et al., 2017, Poggio et al., 2018, Neyshabur et al., 2017, Jiang et al., 2019, 2020]. I will combine theoretical tools from PAC-Bayes, the infinite-width MFT of DNNs, and AIT, with an experiment approach, to develop a consistent picture of why DNNs generalize.

Generalization is linked to *inductive bias*. A learning algorithm usually tries to find a function that fits the data (referred to as *empirical risk minimization*). Any further mechanism which makes choosing one function more likely than other is called an inductive bias². Statistical learning theory establishes that an inductive bias is necessary for generalization, which can be illustrated by considering what would happen in the absence of any inductive bias. An algorithm that chooses any function that fits the data (from the set of all possible functions³) with equal probability, will have very poor generalization, as its output on any new example is equivalent to a uniform random guess.

In Valle-Pérez et al. [2018], we introduced the notion of the *parameter-function map* (PF map) defined as the map between the parameters of a neural network and the function that the neural network with those parameters expresses. It is formally defined in Section 2.1. We empirically found that this map showed a bias towards simple functions, of the same form as the *simplicity bias* found in other maps [Dingle et al., 2018a]. We suggest the following hypothesis: that this bias may be the main source of inductive bias for DNNs, and support

¹PAC learning guarantees generalization error less than a certain value, with at least a certain probability, under samples of the training set. See Section 4.2.2 and Shalev-Shwartz and Ben-David [2014]

²A general learning algorithm may not have a clean distinction between a “fitting the data” component and an “inductive bias” component. Bayesian algorithms, however, do offer such a clean distinction: “fitting the data” is formally identified with the likelihood part of the posterior and the inductive bias is identified with the prior.

³Note that in this work a restriction to the hypothesis class is referred to as an inductive bias too.

this with other experiments comparing the PF map bias of DNN architectures with different generalization.

To further investigate the hypothesis, we adopt the MFT of DNNs, also known as Neural Network Gaussian processes (NNGPs). This theory allows one to perform Bayesian inference with models which are equivalent to infinite-width DNNs, and with a i.i.d. prior $P(\theta)$ for the parameters of the network (typically Guassian). We summarize the MFT of DNNs and NNGPs in Chapter B. We argue that because this prior is essentially uniform in high dimensions, the prior on function space $P(f)$ is mainly determined by the PF map, and empirically show that the function space prior is not very sensitive to the choice of parameter space prior.

The literature on NNGPs has shown that NNGPs show similar generalization performance to their corresponding finite-width DNNs trained with stochastic gradient descent (SGD). This implies that the inductive bias in $P(f)$ is *enough* to explain the generalization of DNNs. In Valle-Pérez et al. [2018], we hypothesized that this was because SGD may be sampling the parameter space close to uniformly (and thus close to Bayesian inference), and we gave extensive evidence in support of this hypothesis in Mingard et al. [2020]. This offers empirical justification for studying Bayesian DNNs, even though most DNNs in practice are trained with SGD (or variants thereof).

In Valle-Pérez et al. [2018], we also applied PAC-Bayes theory with the NNGP prior, to connect the generalization error to the Bayesian evidence (as originally found by Mackay [1992a]). This offers quantitative insight into the statistical origins of generalization in DNNs, and allows us to estimate the error using only the training set. A method for predicting generalization based only on the training set allows one to train the algorithm on all data available, as no test set is required to estimate performance. Using all the data is particularly useful when the amount of data is small. Therefore the PAC-Bayes prediction may be useful when performing model selection, or neural architecture search [], on certain low-data regimes. However, we think a more promising direction is the study of learning curves, as we will see next. Learning curves is the name given to the behaviour of the average generalization error versus the number of samples, where the average is typically over draws of the training set.

In Valle-Perez and Louis, we extended the PAC-Bayes theorem to give high-probability predictions to the actual error, rather than the expected error (see Section 4.4), and we prove

a result that shows that under some weak conditions, the Bayesian evidence should display a ‘learning curve’ with the same exponent as the true learning curve. We experimentally demonstrate this over a large range of modern DNN architectures and datasets, demonstrating that our mean field PAC-Bayes theory has the best predictive power for generalization error of the different proposed theories. We also discuss its limitations, which come mainly from its use of some uncontrolled approximations, and the significant computational expense in computing the bound for large datasets sizes.

The experiments of Mingard et al. [2020] and Valle-Perez and Louis together offer substantial evidence for the hypothesis originally proposed in Valle-Pérez et al. [2018] that the PF map is the main source of inductive bias of DNNs, and represent one of the main results of this thesis. On the theoretical side, and building on this hypothesis, we offer a quantitative framework for predicting generalization and learning curves based on extensions to the PAC-Bayes theory for DNNs. We also analyzed some intriguing properties of the PF map of the simplest neural network, the perceptron, proving that it has a very specific form of bias towards low entropy functions [Mingard et al., 2019]. In Valle-Pérez et al. [2018], we also considered the deeper question of why the inductive bias leads to generalization. Fundamentally, generalization is expected when the inductive bias is similar to the target functions which we want the networks to learn (an statement made rigorous by the PAC-Bayes theorem). Characterizing the distribution of target functions found in real-world problems is a difficult task, but it is generally agreed that problems on which AI is applied are “simple” or “structured” in a way that can be at least approximately captured by the notion of Kolmogorov complexity [Schmidhuber, 1997, Bengio et al., 2007, Lin et al., 2017]. Therefore, we argue, that the inductive bias of the PF map is good because it is also biased towards simple functions.

The thesis is organized as follows. In Chapter 2, we show that the PF map is biased towards simple functions for a simple DNN implementing Boolean functions. We explore the effect of changing the complexity measure, parameter distribution and number of layers. We also link the simplicity bias to some effects that the target function complexity has on learning. In Chapter 3, we cover the theoretical and experimental results about the PF map of the perceptron. We prove that it is biased towards low entropy (high class imbalance) functions, and extend the result to infinite-width multilayer perceptrons with ReLU nonlinearity.

We also show empirical results on the effect of bias. In Chapter 4, we frame existing works on generalization in deep learning under a unified perspective, propose our new PAC-Bayes mean-field theory of generalization, and prove some theoretical results regarding its tightness, based on the asymptotic behaviour of learning curves. In Chapter 5, we present the results from extensive experiments testing the predictive power of the PAC-Bayes theory on a large range of architectures and datasets. In Chapter 6, we present extensive empirical evidence for the applicability of our theory by showing that DNNs trained with standard optimization algorithms have a similar inductive bias to Bayesian neural networks. Finally, in Chapter 7, the significance of the results and future directions are discussed.

Chapter 2

Simplicity bias in the parameter-function map

We investigate the bias of the parameter-function (PF) map. A biased PF map is one where the regions of parameter space mapping to different functions vary a lot in size, so that, intuitively, uniform sampling of parameters will produce a very non-uniform sampling of functions. We will find that the PF map of DNNs is in fact biased towards simple functions.

The huge bias in the PF map we find here for DNNs is similar to that found for networks of Boolean units [Van den Broeck and Kawai, 1990]. In Carnevali and Patarnello [1987], it was argued that generalization in Boolean networks occurs for certain problems because there are many networks which produce those problems, so that the optimization algorithm is likely to choose one of them, even when constrained only by few training examples. In this thesis we will argue that the same fundamental idea holds for modern neural networks. In Carnevali and Patarnello [1987] and [Van den Broeck and Kawai, 1990] they also observed, in the few examples they tried, that simple Boolean functions tended to have a higher a priori probability. In this chapter we will establish this observation for simple DNNs with extensive experiments where we will measure the complexity of functions, using different complexity measures. We offer a partial explanation of the simplicity bias based on AIT [Dingle et al., 2018b]. We will also explore the effect of some simple architecture changes, such as depth.

In this chapter, we use intuitive arguments to suggest that the simplicity bias leads to good generalization for simple functions and bad generalization for complex functions, which was

observed in Franco [2006] and more recently in Zhang et al. [2017a]. Our argument is that if the optimizer samples functions similarly to Bayesian inference, then we would expect better generalization for target functions with higher probability as Carnevali and Patarnello [1987] argued. The simplicity bias, then implies the better generalization for simple functions. In later chapters we will develop a quantitative theory for this intuition, which we will see differs significantly from other work on theories of generalization for deep learning (Chapter 4).

2.1 Deep neural networks and the parameter-function map

We begin by defining the main class of models we will be studying, fully connected feed-forward neural networks (FCN). These are the simplest type of neural network, and serves as a starting point of most theoretical analyses. We will look at more complicated architectures later on, but this definition will help set our notation for the thesis.

Definition 2.1.1 (DNNs). *Fully connected feed-forward neural networks with activations σ and a single output neuron form a parameterised function family $f(x)$ on inputs $x \in \mathbb{R}^n$. This can be defined recursively, for L hidden layers, $1 \leq l \leq L$, as*

$$\begin{aligned} f(x) &= H(h^{(L+1)}(x)), \\ h^{(l+1)}(x) &= w_l \sigma(h^{(l)}) + b_l, \\ h^{(1)}(x) &= w_0 x + b_0, \end{aligned}$$

where $H(X)$ is the Heaviside step function defined as 1 if $X > 0$ and 0 otherwise, and σ is an activation function that acts element-wise. The $w_l \in \mathbb{R}^{n_{l+1} \times n_l}$ are the weights, and $b_l \in \mathbb{R}^{n_{l+1}}$ are the threshold bias weights at layer l , where n_l is the number of hidden neurons in the l -th layer. n_{L+1} is the number of outputs (1 in this thesis), and n_0 is the dimension of the inputs (which we will also refer to as n).

We will refer to the whole set of parameters (w_l and b_l , $1 \leq l \leq L$) as θ . We define the parameter-function map as in [Valle-Pérez et al., 2018] below.

Definition 2.1.2. (*Parameter-function map*) For a parametrized supervised learning model, let the input space be \mathcal{X} and the output space be \mathcal{Y} . The space of functions that the model can express is then $\mathcal{F} \subseteq \mathcal{Y}^{|\mathcal{X}|}$. If the model has p real valued parameters, taking values within a set $\Theta \subseteq \mathbb{R}^p$, the parameter-function map, \mathcal{M} , is defined as:

$$\begin{aligned}\mathcal{M} : \Theta &\rightarrow \mathcal{F} \\ \theta &\mapsto f_\theta\end{aligned}$$

where f_θ is the function implemented by the model with choice of parameter vector θ .

This map is of interest when using an algorithm that searches in parameter space, such as stochastic gradient descent (SGD), as it determines how the behaviour of the algorithm in parameter space maps to its behavior in function space. The latter is what determines many properties of interest such as generalization.

2.2 Algorithmic information theory and simplicity-bias in the parameter-function map

One of the first sources of inspiration for this work comes from a paper [Dingle et al., 2018a] which finds that “simple” maps, or more generally “simple” probability distributions, have the property that any sufficiently strong bias must be towards simple outputs. Simplicity is defined by the formal notion of *Kolmogorov complexity*, which measures the size of the smallest program producing a particular object that can be encoded as a sequence of symbols.

Specifically, Dingle et al. [2018a] derives the following upper bound for the probability $P(x)$ that an output $x \in O$ of an input-output map $g : I \rightarrow O$ obtains upon random sampling¹ of inputs I

$$P(x) \leq 2^{-a\tilde{K}(x)+b} \tag{2.1}$$

¹We typically assume uniform sampling of inputs, but other distributions with simple descriptive complexity would also work

where $\tilde{K}(x)$ is an approximation to the (uncomputable) Kolmogorov complexity of x , and a and b are scalar parameters which depend on the map g but not on x . This bound has some technical requirements. The main one is that the input-output map must itself have low Kolmogorov complexity. Specifically, a computable² input-output map $f : I \rightarrow O$, mapping N_I inputs from the set I to N_O outputs x from the set O ³ may exhibit simplicity bias if the following restrictions are satisfied (Dingle et al. [2018a]):

- 1) *Map simplicity*: The map should have limited complexity, that is its Kolmogorov complexity $K(f)$ should asymptotically satisfy $K(f) + K(n) \ll K(x) + O(1)$, for typical $x \in O$ where n is a measure of the size of the input set (e.g. for binary input sequences, $N_I = 2^n$). This is the main condition for Equation (2.1) to formally hold.
- 2) *Redundancy*: There should be many more inputs than outputs ($N_I \gg N_O$) so that the probability $P(x)$ that the map generates output x upon random selection of inputs $\in I$ can in principle vary significantly.
- 3) *Finite size* $N_O \gg 1$ to avoid potential finite size effects.
- 4) *Nonlinearity*: The map f must be a nonlinear function since linear functions do not exhibit bias.
- 5) *Well behaved*: The map should not primarily produce pseudorandom outputs (such as the digits of π), because complexity approximators needed for practical applications will mistakenly label these as highly complex.

For the deep learning learning systems studied in this paper, the inputs of the map f are the parameters that fix the weights for the particular neural network architecture chosen, and the outputs are the functions that the system produces. Consider, for example, DNNs producing Boolean functions, like we study in the next section. While the output functions rapidly grow in complexity with increasing size of the input layer, the map itself can be described with a low-complexity procedure, since it consists of reading the list of parameters, populating a given neural network architecture and evaluating it for all inputs. For reasonable architectures,

²Here computable simply means that all inputs lead to outputs, in other words there is no halting problem.

³This language of finite input and outputs sets assumes discrete inputs and outputs, either because they are intrinsically discrete, or because they can be made discrete by a coarse-graining procedure. For the parameter-function maps studied in this paper the set of outputs (the full hypothesis class) is typically naturally discrete, but the inputs are continuous. However, the input parameters can always be discretised without any loss of generality.

the information needed to describe the parameter-function map grows logarithmically with the input dimension n , so for large enough n , the amount of information required to describe the map will be much less than the information needed to describe a typical function, which requires 2^{2^n} bits. Thus the Kolmogorov complexity $K(f)$ of this map is asymptotically smaller than the typical complexity of the output, as required by the map simplicity condition 1) above.

The redundancy condition 2) depends on the network architecture and discretization. For overparameterised networks, this condition is typically satisfied. In the case we study in this chapter, where we use floating point numbers for the parameters (input set I), and Boolean functions (output set O), this condition is clearly satisfied. Neural nets can represent very large numbers of potential functions (see for example estimates of VC dimension Harvey et al. [2017], Baum and Haussler [1989]), so that condition 3) is also generally satisfied. Neural network parameter-function maps are evidently non-linear, satisfying condition 4). Condition 5) is perhaps the least understood condition within the study of simplicity bias. However, in the experiments in the next section, we will find no function with high probability and high complexity (at least when using LZ complexity), which provides some empirical validation. This condition also agrees with the expectation that neural networks will not predict the outputs of a good pseudorandom number generator. One of the implicit assumptions in the simplicity bias framework is that, although true Kolmogorov complexity is always uncomputable, approximations based on well chosen complexity measures perform well for most relevant outputs x . Nevertheless, where and when this assumption holds is a deep problem for which further research is needed.

If there is a significant linear variation in $K(x)$, then the bound (2.1) will vary over many orders of magnitude. While the theoretical arguments in Dingle et al. [2018a] do not prove that specific maps will be biased, they do say that if there is sufficient bias, the bias will be towards simple outputs, so that it satisfies Eq. (2.1). Empirically, it was shown in Dingle et al. [2018a] that this bound works well for many input-output maps ranging from the RNA sequence to secondary structure map to a map for option pricing in financial mathematics. Furthermore, many of these maps were found to be sufficiently biased that the $P(x)$ was typically close to the bound. In this case $P(x) \sim 2^{-a\tilde{K}(x)+b}$ can offer a relatively good approximation, and we

refer to those maps, informally, as *simplicity-biased*.

As we saw above, the DNN parameter-function map \mathcal{M} is simple in the sense required by the bound, and also fulfills the other necessary conditions for simplicity bias. The success of Eq. (2.1) in predicting the behaviour for other maps suggests that the parameter-function map of many different kinds of DNNs could also exhibit simplicity bias. This observation motivated the experiments carried out in the next section, and led to the discovery that indeed the PF map of neural networks is simplicity-biased.

2.3 Empirical study of simplicity bias in a DNN implementing Boolean functions

In order to explore the properties of the parameter-function map of DNNs and whether it is simplicity biased, we consider *random neural networks*. We put a probability distribution $P_{\text{par}}(\theta)$ over the space of parameters Θ , and are interested in the distribution over functions $P(f)$ induced by this distribution via the parameter-function map \mathcal{M} of a given neural network architecture. We call $P(f)$ the *a-priori* or *prior* distribution (see Section B.1), defined as follows

$$P(f) = \tilde{P}(\mathcal{M}^{-1}(f)) \quad (2.2)$$

where $\mathcal{M}^{-1}(f)$ is the preimage of f under the parameter-function map. This is just the total probability mass of the set of parameters $\mathcal{M}^{-1}(f) \subseteq \Theta$ producing the function f . In the experiments in this section, we estimate the probability of different functions by taking a large sample of parameters and simply counting the number of samples producing individual functions (*empirical frequency*).

This procedure is easiest for a discrete space of functions, and only feasible for a small enough function space where the probabilities of obtaining functions more than once is not negligible. We achieve this by using a small neural network with 7 Boolean inputs, two hidden layers of 40 ReLU neurons each, and a single Boolean output. This means that the input space is $\mathcal{X} = \{0, 1\}^7$ and the space of functions is $\mathcal{F} \subseteq \{0, 1\}^{2^7}$ (we checked that this neural network

can in fact express almost all functions in $\{0, 1\}^{2^7}$ ⁴. For the distributions on parameter space we used Gaussian distributions or uniform within a hypercube, and tried several variances.⁵

The resulting probabilities $P(f)$ can be seen in Figure 2.1a, where we plot the (normalized) empirical frequencies versus the rank (when ranked by the frequency), which exhibits a range of probabilities spanning as many orders of magnitude as the finite sample size allows. Using different distributions over parameters has a relatively small effect on the overall curve.

We show in Figure 2.1a that the rank plot of $P(f)$ can be accurately fit with a normalized Zipf law $P(r) = (\ln(N_O)r)^{-1}$, where r is the rank, and $N_O = 2^{2^7} = 2^{128}$. It is remarkable that this form fits the rank plot so well without any adjustable parameters, suggesting that Zipf behaviour holds over the whole rank plot, leading to an estimated range of 39 orders of magnitude in $P(f)$ ⁶. As this is obtained for all the different non-informative parameter-space distributions we tried (see also Section D.4), we conclude that the PF map is extremely biased. Independently, Eq. (2.1) and the results from Dingle et al. [2018a] suggests that to first order $P(f)$ is independent of the number of parameters since it follows from $K(f)$, and the map only enters through a and b , which are constrained (Dingle et al. [2018a]). Also, once the DNN is sufficiently expressive so that all N_O functions can be found, this argument suggests that $P(f)$ will not vary much with an increase in parameters, which we expect to happen eventually from the asymptotic results of the MFT (Chapter B).

The bound of equation (2.1) predicts that high probability functions should be of low descriptive complexity. In Figure D.2, we show that we indeed find this simplicity-bias phenomenon, with all high-probability functions having low Lempel-Ziv (LZ) complexity (defined in Section D.1.1). Furthermore, because of the large bias, a lot of functions have probabilities close to the upper bound, showing the simplicity bias phenomenon described in the previous section.

In Figure 2.2, we show the same data as in Figure 2.1b, but where we weight each function that falls within a region in the probability-complexity plane, by their probability. As a result, we can visualize the probability of obtaining points in different parts of the plane, upon randomly sampling the parameters of the network. We see that most of the weight

⁴Note that although this system is small, there are still $N_O = \approx 3 \times 10^{38}$ possible functions.

⁵for Figures 2.1b, 2.2 we used a uniform distribution with variance of $1/\sqrt{n}$ with n the input size to the layer

⁶We have tried to figure out an explanation of this Zipfian behaviour but so far without success

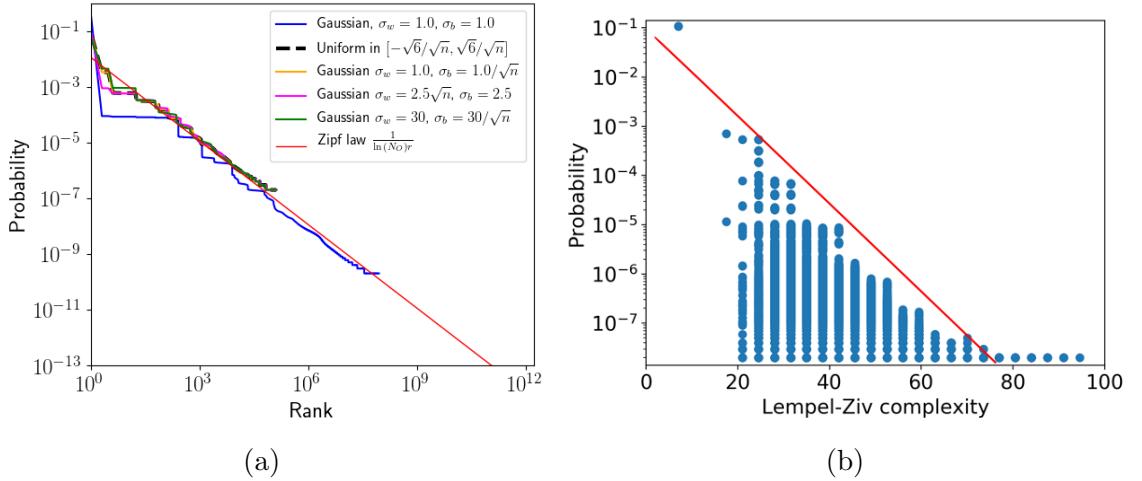


Figure 2.1: (a) Probability versus rank of each of the functions (ranked by probability) from a sample of 10^{10} (blue) or 10^7 (others) parameters. The labels are different parameter distributions. (b) Probability versus Lempel-Ziv complexity. Probabilities are estimated from a sample of 10^8 parameters. Points with a frequency of 10^{-8} are removed for clarity because these suffer from finite-size effects. The red line is the simplicity bias bound of Equation (2.1)

(d) Generalization error increases with the Lempel-Ziv complexity of different target functions, when training the network with SGD.

falls close to the upper bound throughout all the orders of magnitude of probability or values complexity, as was also found for other maps in Dingle et al. [2018a]. This confirms that $P \sim 2^{-aK+b}$ is a good approximation, and that the map is indeed simplicity-biased.

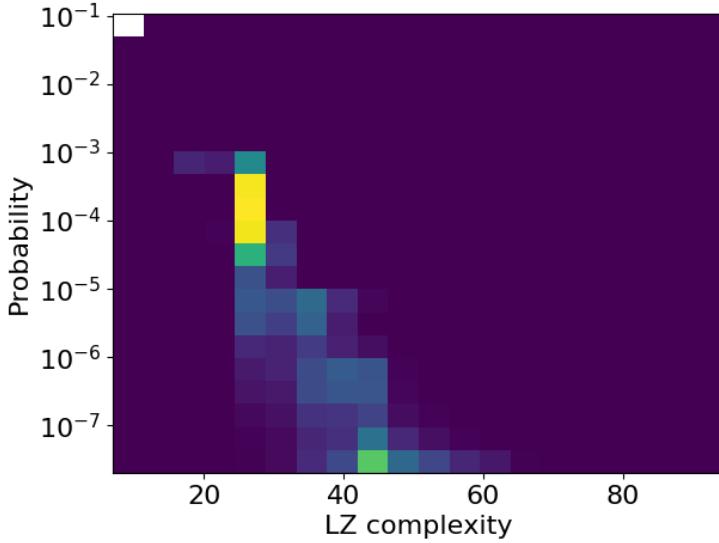


Figure 2.2: Histogram of functions in the probability versus Lempel-Ziv complexity plane, weighted according to their probability. Probabilities are estimated from a sample of 10^8 parameters, for a network of shape $(7, 40, 40, 1)$

The literature on complexity measures is vast. Here we simply note that there is nothing fundamental about LZ. Other approximate complexity measures that capture essential aspects of Kolmogorov complexity also show similar correlations. In Section D.1, we show how we obtain similar results for other complexity measures, and comment on their differences.

There are many reasons to believe that real-world functions are simple or have some structure (Lin et al. [2017], Schmidhuber [1997]) and will therefore have low descriptive complexity. Putting this together with the above results means we expect a DNN that exhibits simplicity bias to generalize well for real-world datasets and functions. On the other hand, in Section 2.4, we show that our network does not generalize well for complex (random) functions, which can be linked to the No Free Lunch theorem. By simple counting arguments, the number of high complexity functions is exponentially larger than the number of low complexity functions. Nevertheless, such functions may be less common in real-world applications.

2.4 Effects of target function complexity on learning

Here we show the effect of the complexity of the target function on learning, as well as other complementary results. As discussed in Section 2.3, a consequence of simplicity bias is DNNs

can generalize if the solution is simple. On the other hand, the No Free Lunch theorem [Wolpert and Waters, 1994] states that no algorithm can generalize better (for off-training error) uniformly over all functions than any other algorithm, so that DNNs must still fail to generalize for complex functions (which is the vast majority of functions under uniform sampling).

In our experiments we look at the following learning metrics: the distance travelled in parameter space, the number of iterations to reach 0 training error, and the complexity of the learned function. We also compare the generalization of neural networks to random guessing a function from the set of all possible Boolean functions, which we call “unbiased learner”. Note this is a fair comparison because the neural networks we use have a hypothesis class that is at least very close to the set of all Boolean functions, as we tested that the neural network used here can perfectly fit a big sample of random functions.

The target functions in these experiments were chosen by randomly sampling parameters of the neural network used, and so even the highest complexity ones are probably not fully random⁷. In fact, when training the network on truly random functions, we obtain generalization errors equal or above those of the unbiased learner, as expected from the NFL theorem.

In Figure 2.3, we show the different learning metrics versus the LZ complexity. In Figures D.4 to D.6 in Section D.3, we show the results for generalization complexity, Boolean complexity, and entropy, of the target function, respectively, where we see that all the complexity measures show very similar qualitative behaviour. We see that the generaliation error grows with the complexity of the target, as expected. The complexity of the function learned by the neural network also grows with the target function complexity. This can be understood for algorithms that prefer simple functions, as the complexity of the simplest function that fits the data grows with the target function complexity. We analyze this phenomenon further in Chapter H.

The number of iterations and distance travelled in parameter space also grow, which may be attributed to simplicity bias as the regions of parameter space corresponding to simple functions should be larger, and thus probably easier to find by optimizers in fewer steps and

⁷The fact that non-random strings can have maximum LZ complexity is a consequence of LZ complexity being a less powerful complexity measure than Kolmogorov complexity, see e.g. Estevez-Rams et al. [2013]. The fact that neural networks do well for non-random functions, even if they have maximum LZ, suggests that their simplicity bias captures a notion of complexity stronger than LZ, or at least that it is able to capture some regularity that LZ cannot capture.

with less distance travelled. This is analogous to the arguments for the “arrival of the frequent” in evolution [Schaper and Louis, 2014], where the authors argue that populations may evolve towards some phenotypes because they are found *quicker*. This can also be related to the result we mentioned above about the complexity of the simplest function fitting the data. we may expect that for most points on whicht the optimizer is initialized, regions of parameter space corresponding to complex functions, are further away than those corresponding to simple functions, as the latter have larger regions of parameter space mapping to them. Therefore, if we assume that the optimizer will tend to find the closest parameters that fit the data, as the complexity of the simplest function fitting the data grows with the target function complexity, we may expect that the distance that needs to be travelled to fit the data also grows.

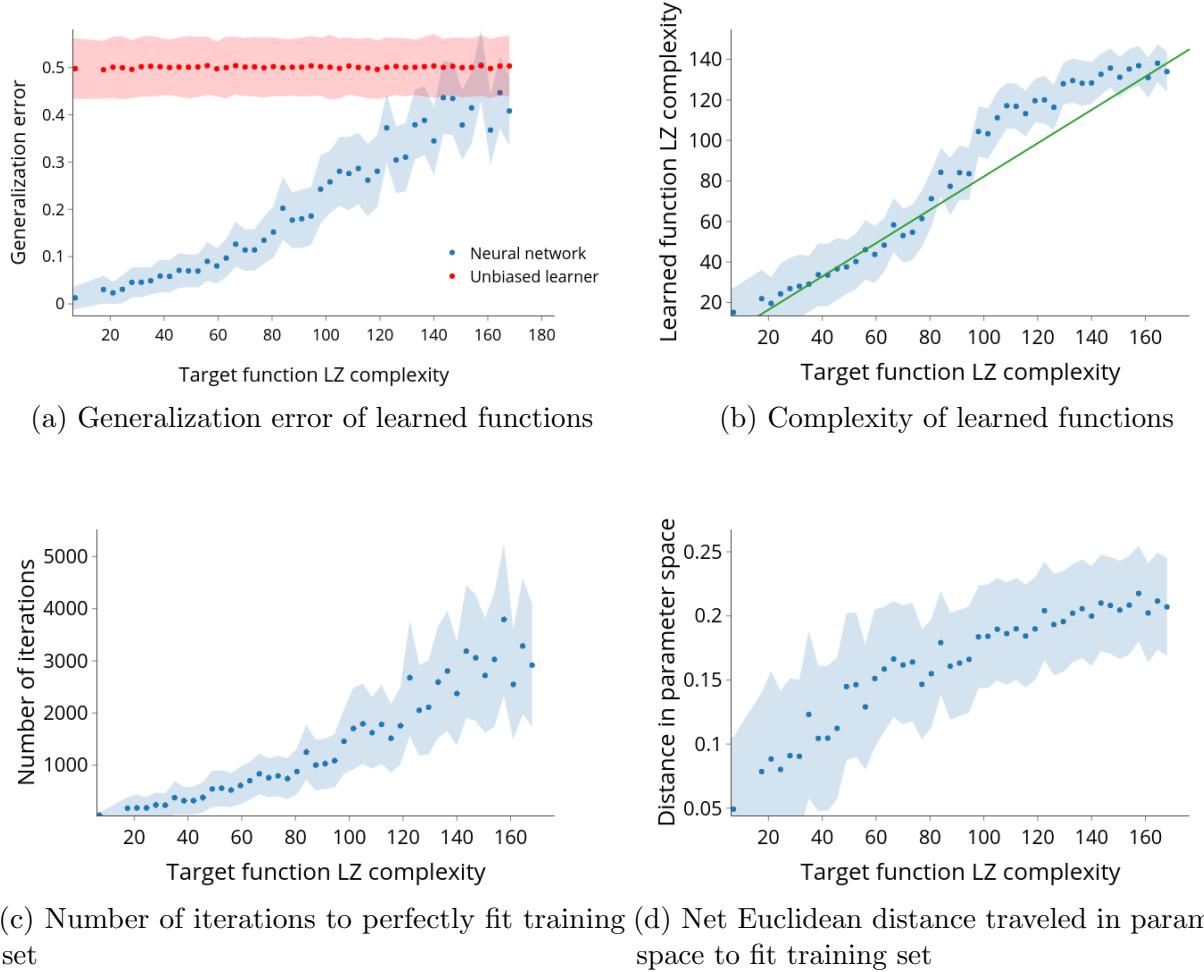


Figure 2.3: Different learning metrics versus the LZ complexity of the target function, when learning with a network of shape $(7, 40, 40, 1)$. Dots represent the means, while the shaded envelope corresponds to piecewise linear interpolation of the standard deviation, over 500 random initializations and training sets.

2.4.1 Lempel-Ziv versus Entropy

To check that the correlation between LZ complexity and generalization is not only because of a correlation with function entropy (which is just a measure of the fraction of inputs mapping to 1 or 0, see Section D.1.1, and thus fails to capture many types of regularity), we observed that for some target functions with maximum entropy (but which are simple when measured using LZ complexity), the network still generalizes better than the unbiased learner, showing that the bias towards simpler functions is better captured by more powerful complexity measures than

entropy⁸. This is confirmed further by the results in Fig. 2.4 where we fix the target function entropy to 1.0, and observe that the generalization error still exhibits considerable variation, as well as a positive correlation with LZ complexity.

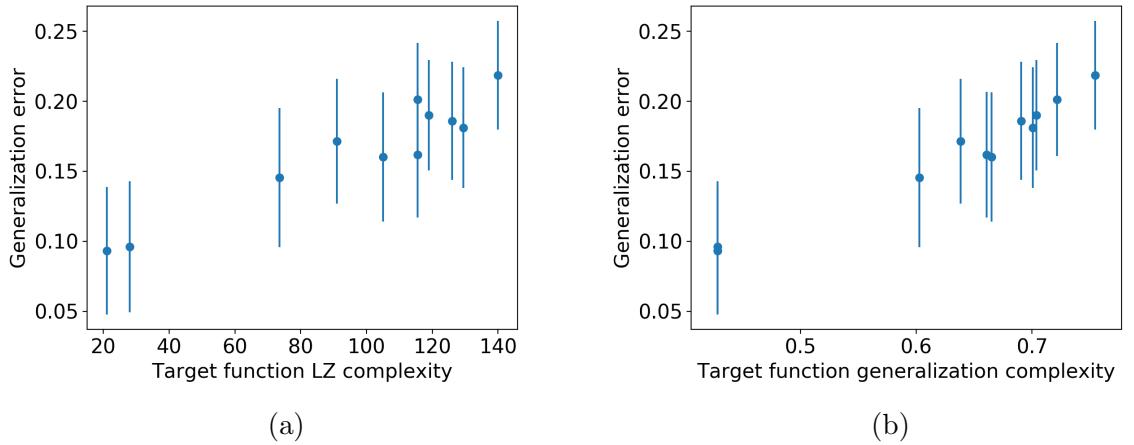


Figure 2.4: Generalization error of learned function versus the complexity of the target function for target functions with fixed entropy 1.0, for a network of shape $(7, 20, 20, 1)$. Complexity measures are (a) LZ and (b) generalisation complexity. Here the training set size was of size 64, but sampled with replacement, and the generalization error is over the whole input space. Note that despite the fixed entropy there is still variation in generalization error, which correlates with the complexity of the function. These figures demonstrate that entropy is a less accurate complexity measure than LZ or generalisation complexity, for predicting generalization performance.

2.5 Simplicity bias in larger systems

Although here we have only shown that high-probability functions have low complexity for a relatively small DNN, the generality of the AIT arguments from Dingle et al. [2018a] suggests that an exponential probability-complexity bias should hold for larger neural networks as well. In this section, we look at some ways to explore simplicity bias in larger DNNs and on more realistic datasets. Although we will be looking at DNNs with a single Boolean output, larger input dimensions quickly make direct estimation of function probabilities unfeasible (as the number of functions grows as a double exponential with input dimensionality). Furthermore,

⁸LZ is a better approximation to Kolmogorov complexity than entropy (Cover and Thomas [2012]), but of course LZ can still fail, for example when measuring the complexity of the digits of π .

as we cannot represent the functions as a complete truth table anymore, we cannot use the complexity measures we used for Boolean functions.

To deal with the second problem, we will use the original version of the critical sample ratio (CSR) defined for continuous inputs, which measures the number of inputs which are near the classification boundary Krueger et al. [2017a] (see Section D.1.1 for the full definition). To deal with the first problem, we take two approaches. First, we look at the histogram of complexity values as a more coarse-grained way to look for simplicity bias. Second, we use the MFT of DNNs (NNGPs) to estimate the probability of labellings, without need for expensive sampling.

In the first set of experiments, we will look at the complexity histogram of two CNNs which show different generalization performance. In ref. Ba and Caruana [2014], it is demonstrated that certain shallow neural networks with sufficient number of parameters can learn to mimic a deeper network and reach comparable accuracy in an image classification task using the CIFAR10 dataset, demonstrating that the shallow network has enough expressivity to solve this task. However, when training directly on the training data, the deep network succeeds in finding a function with high generalization accuracy, while the shallow network fails. We explored whether differences in the simplicity bias in the parameter-function map could be the cause of this difference. In Figure 2.5 we see that, measuring complexity using CSR, when randomly sampling parameters the deeper network produces simple functions much more often than the shallow function, demonstrating a stronger simplicity bias in the parameter-function map of the deep network. This could explain why the deep network generalizes better when trained on the CIFAR10 data, even though both networks have enough expressive power.

Note that although for the network in Figure 2.5a a complex function is more likely than a simple function, this is still compatible with the simplicity bias in the parameter-function map. As there are typically exponentially more complex functions than simple ones, the histogram of complexities for an unbiased parameter-function map would be much more skewed toward complex functions. Therefore, the results in Figure 2.5 just tell us that the shallow network has significantly *less* simplicity bias than the deep network, not that it doesn't have simplicity bias. In Figure D.3 in Section D.2, we provided a similar analysis for the Boolean system networks, which also shows that a shallow network is less biased than a deep one.

To gain a finer-grained picture into the simplicity bias of these larger networks, we will

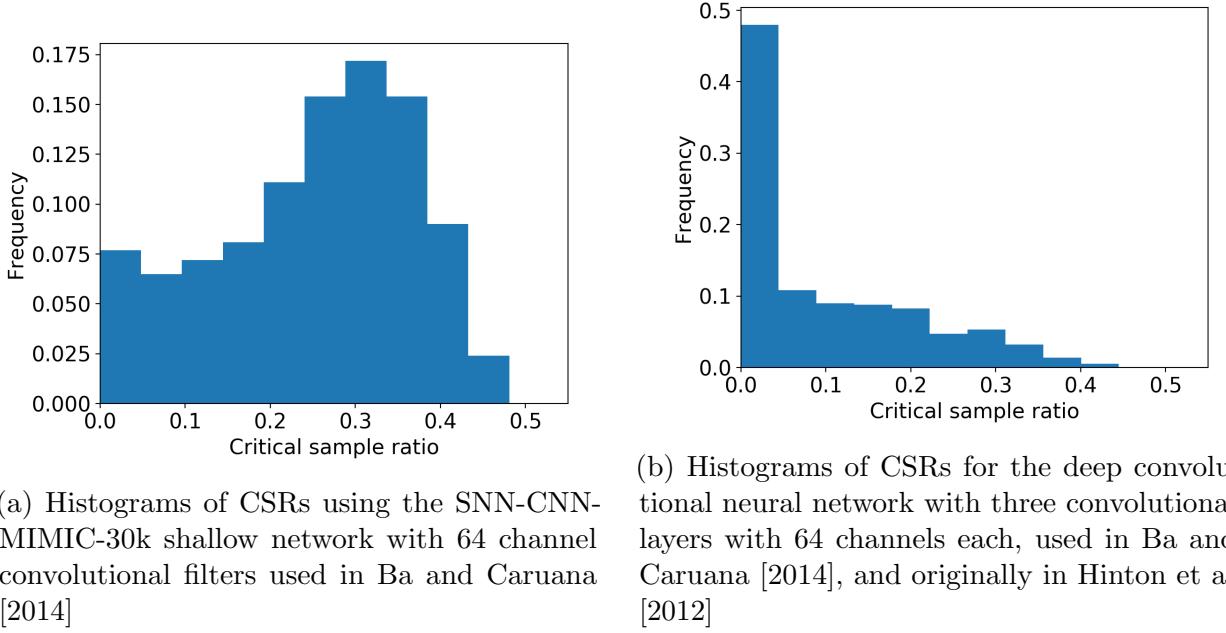


Figure 2.5: Normalized histograms of critical sample ratio (CSR) estimated from a sample of 1000 images from the CIFAR10 dataset. These are obtained by randomly sampling parameters (using a Gaussian distribution with variance as in Xavier initialization). The parameter sample size is also 1000.

now use the NNGP approximation to estimate probabilities of different labelings on a random sample of 1000 images from CIFAR10. We use a 4-layer CNN defined in Section A.1, and randomly sample parameters to obtain a set of labellings. We then calculate average CSR of the sampled networks corresponding to different labellings (note that CSR depends on the whole classification surface, and not just on the labelling of a finite set of points). Finally, we plot this average CSR versus the probability of the labelling as estimated by the corresponding NNGP. As can be seen in Figure 2.6 they correlate remarkably well, suggesting that simplicity bias is also found for more realistic DNNs.

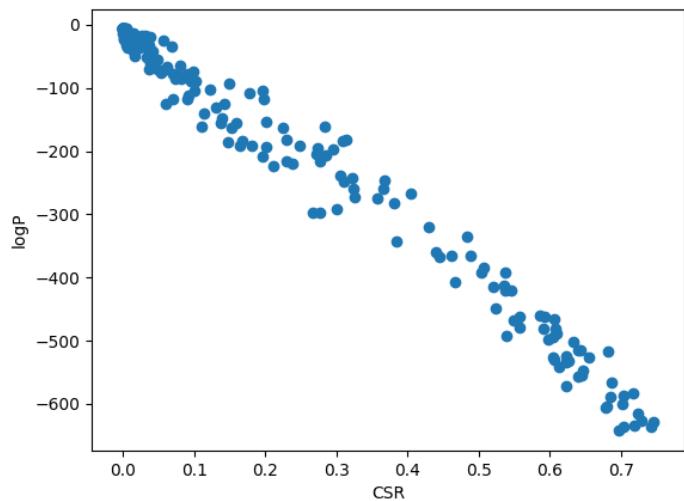


Figure 2.6: Probability (using GP approximation) versus critical sample ratio (CSR) of labelings of 1000 random CIFAR10 inputs, produced by 250 random samples of parameters. The network is a 4 layer CNN.

Chapter 3

The perceptron parameter-function map

A full theoretical characterization of the simplicity bias in the PF map, observed in Chapter 2, is still an open research question. Some steps have been done in this regard. Most work has looked at the PF map of infinitely-wide neural networks (MFT limit) with Boolean inputs: De Palma et al. prove that it is biased towards functions which are robust to perturbations of their inputs (low input sensitivity), while Yang and Salman [2019] show that it exhibits a certain kind of bias towards Boolean functions expressible with low-degree monomials.

To gain further insight into the nature of the bias of the PF map, in this chapter we study the simplest kind of neural network, the perceptron. We will also extend some of the results to infinitely-wide feedforward ReLU DNNs. As in Chapter 2, we study how likely different Boolean functions, defined as $f : \{0, 1\}^n \rightarrow \{0, 1\}$, are obtained upon randomly chosen weights of neural networks. This corresponds to the prior distribution over functions $P(f)$, upon random initialization of the parameters, which reflects the inductive bias of training algorithms that approximate Bayesian inference. As we empirically show in Chapter 6, many optimizers commonly used in deep learning approximate Bayesian inference.

We will also focus on one of the simplest measures of complexity, namely the entropy. As we saw in Chapter 2, the entropy $H(f)$ of a Boolean function f is defined as the binary entropy of the fraction of possible inputs to f that f maps to 1. This quantity essentially measures the amount of class imbalance of the function, and is complementary to previous works studying

notions of smoothness as a proxy for complexity.

3.1 Summary of key results

The work in this chapter was done in collaboration with Chris Mingard et al. Mingard et al. [2019]. The idea of the paper is due to Chris M., Joar S. and myself. The majority of the experiments were done by Chris M., supervised by Ard AL. and myself, with technical help from Vladimir M. The proof of the main theorems (Theorem 3.3.1 and Theorem 3.4.5) were done by me and David MR., based on a lot of helpful ideas and discussions with Chris M. The results on expressivity (Section 3.4.1) were due to Joar S. I also conceptualized and did the learning experiments in Section 3.7. The main results are the following

1. In Section 3.3 we study a simple perceptron with no threshold bias term, and with weights w sampled from a distribution which is symmetric under reflections along the coordinate axes. Let the random variable T correspond to the number of points in $\{0, 1\}^n$ which fall above the decision boundary of the network (i.e. $T = |\{x \in \{0, 1\}^n : \langle w, x \rangle > 0\}|$) upon i.i.d. random initialisation of the weights. We prove that T is distributed uniformly, i.e. $P(T = t) = 2^{-n}$ for $0 \leq t < 2^n$. Let \mathbb{F}_t be the set of all functions with $T = t$ that the perceptron can produce and let $|\mathbb{F}_t|$ be its size (cf. Theorem 3.2.2). We expect $|\mathbb{F}_t|$ for $t \sim 2^{n-1}$ (high entropy) to be (much) larger than $|\mathbb{F}_t|$ for extreme values of t (low entropy). The average probability of obtaining a particular function f which maps t inputs to 1 is $2^{-n}/|\mathbb{F}_t|$. The perceptron therefore shows a strong bias towards functions with low entropy, in the sense that individual functions with low entropy have, on average, higher probability than individual functions with high entropy.
2. In Section 3.3.3, we show that within the sets \mathbb{F}_t , there is a further bias, and in some cases this is clearly towards simple functions which correlates with Lempel-Ziv complexity [Lempel and Ziv, 1976, Dingle et al., 2018c], as predicted in [Valle-Pérez et al., 2018].
3. In Section 3.3.4, we show that adding a threshold bias term to a perceptron significantly increases the bias towards low entropy.

4. In Section 3.4.1, we provide a new expressivity bound for Boolean functions: DNNs with input size n , l hidden layers each with width $n + 2^{n-1-\log_2 l} + 1$ and a single output neuron can express all 2^{2^n} Boolean functions over n variables.
5. In Section 3.4.2 we generalise our results to neural networks with multiple layers, proving (in the infinite-width limit) that the bias towards low entropy increases with the number of ReLU-activated layers.
6. In Section E.1, we discuss the connections between the bias towards low entropy and the ordered regime of the infinite-width MFT of DNNs [Poole et al., 2016, Schoenholz et al., 2017], showing that in the chaotic regime the bias disappears.
7. In Section 3.5, we also show some empirical evidence that the results derived in this paper seem to generalize beyond the assumptions of our theoretical analysis, to more complicated data distributions (binarized MNIST and CIFAR10) and architectures (CNNs).
8. Finally, in Section 3.7, we show preliminary results on the effect of entropy-like biases in $P(f)$ on learning class-imbalanced data.

3.2 Definitions, Terminology, and Notation

As we saw in Section 2.1, we will refer to the whole set of parameters (w_l and b_l , $1 \leq l \leq L$) as θ . In the case of perceptrons we use $f_\theta(x) = \sigma(\langle w, x \rangle + b)$ to specify a network.

In this chapter we are interested in the Boolean functions that neural networks express. We consider the 0-1 Boolean hypercube $\{0, 1\}^n$ as the input domain.

Definition 3.2.1. *The function $\mathcal{T}(f)$ is defined as the number of points in the hypercube $\{0, 1\}^n$ that are mapped to 1 by the action of a neural network f .*

For example, for a perceptron this function is defined as,

$$\mathcal{T}(f) = \mathcal{T}(w, b) = \sum_{x \in \{0,1\}^n} \mathbf{1}(\langle x, w \rangle + b). \quad (3.1)$$

We will sometimes use $\mathcal{T}(w, b)$ if the neural network is a perceptron.

Definition 3.2.2 (\mathbb{F}_t and $P(t)$). We define the set \mathbb{F}_t to be the set of functions expressible by some model \mathcal{M} (e.g. a perceptron, a neural network) which all have the same value of $\mathcal{T}(f)$,

$$\mathbb{F}_t = \{f \in \mathcal{F}_{\mathcal{M}} | \mathcal{T}(f) = t\}$$

, where $\mathcal{F}_{\mathcal{M}}$ is the set of all functions expressible by \mathcal{M} . Given a probability measure P on the weights θ , we define the probability measure

$$P(T = t) := P(\theta : f_{\theta} \in \mathbb{F}_t)$$

We can also define $\mathcal{T}(f)$ and $P(T = t)$ in the natural way for sets of input points other than $\{0, 1\}^n$, the context making clear what definition is being used.

Definition 3.2.3. The entropy $H(f)$ of a Boolean function $f : \{0, 1\}^* \rightarrow \{0, 1\}$ is defined as $H(f) = -p \log_2 p - (1 - p) \log_2 (1 - p)$, where $p = \mathcal{T}_f / 2^n$. It is the binary entropy of the fraction p of possible inputs to f that f maps to 1 or equivalently, the binary entropy of the fraction of 1's in the right-hand column of the truth table of f .

Definition 3.2.4. We define the Boolean complexity $K_{\text{Bool}}(f)$ of a function f as the number of binary connectives in the shortest Boolean formula that expresses f .

Note that Boolean complexity can be defined in other ways as well. For example, $K_{\text{Bool}}(f)$ is sometimes defined as the number of connectives (rather than *binary* connectives) in the shortest formula that expresses f , or as the depth of the most shallow formula that expresses f . These definitions tend to give similar values for the complexity of a given function, and so they are largely interchangeable in most contexts. We use the definition above because it makes our calculations easier.

3.3 Intrinsic bias in a perceptron's parameter-function map

In this section we study the parameter-function map of the perceptron, in many ways the simplest neural network. While it famously cannot express many Boolean functions, like XOR

Marvin and Seymour [1969], it remains an important model system. Moreover, many DNN architectures include layers of perceptrons, so understanding this very basic architecture may provide important insight into the more complex neural networks used today.

3.3.1 Entropy bias in a simple perceptron with $b = 0$ (no threshold bias term)

Here we consider perceptrons $f_\theta(x) = \mathbf{1}(\langle w, x \rangle + b)$ without threshold bias terms, i.e. $b = 0$.

The following theorem shows that under certain conditions on the weight distribution, a perceptron with no threshold bias has a uniform $P(\theta : \mathcal{T}(f_\theta) = t)$. The class of weight distributions includes the commonly used isotropic multivariate Gaussian with zero mean, a uniform distribution on a centred cuboid, and many other distributions. The full proof of the theorem is in Section C.1.

Theorem 3.3.1. *For a perceptron f_θ with $b = 0$ and weights w sampled from a distribution which is symmetric under reflections along the coordinate axes, the probability measure $P(\theta : \mathcal{T}(f_\theta) = t)$ is given by*

$$P(\theta : \mathcal{T}(f_\theta) = t) = \begin{cases} 2^{-n} & \text{if } 0 \leq t < 2^n \\ 0 & \text{otherwise} \end{cases}.$$

Proof sketch. We consider the sampling of the normal vector w as a two-step process: we first sample the absolute values of the elements, giving us a vector w_{pos} with positive elements, and then we sample the signs of the elements. Our assumption on the probability distribution implies that each of the 2^n sign assignments is equally probable, each happening with a probability 2^{-n} . The key of the proof is to show that for any w_{pos} , each of the sign assignments gives a distinct value of T (and because there are 2^n possible sign assignments, for any value of T , there is exactly one sign assignment resulting in a normal vector with that value of T). This implies that, provided all sign assignments of any w_{pos} are equally likely, the distribution on T is uniform. \square

A consequence of Theorem 3.3.1 is that the average probability of the perceptron producing

a particular function f with $\mathcal{T}(f) = t$ is given by

$$\langle P(f) \rangle_t = \frac{2^{-n}}{|\mathbb{F}_t|}, \quad (3.2)$$

where \mathbb{F}_t denotes the set of Boolean functions that the perceptron can express which satisfy $\mathcal{T}(f) = t$, and $\langle \cdot \rangle_t$ denotes the average (under uniform measure) over all functions $f \in \mathbb{F}_t$.

We expect $|\mathbb{F}_t|$ to be much smaller for more extreme values of t , as there are fewer distinct possible functions with extreme values of t . This would imply a bias towards low *entropy* functions. By way of an example, $|\mathbb{F}_0| = 1$ and $|\mathbb{F}_1| = n$ (since the only Boolean functions f a perceptron can express which satisfy $\mathcal{T}(f) = 1$ have $f(x) = 1$ for a single one-hot $x \in \{0, 1\}^n$), implying that $\langle P(f) \rangle_0 = 2^{-n}$ and $\langle P(f) \rangle_1 = 2^{-n}/n$.

Nevertheless, the probability of functions within a set \mathbb{F}_t is unlikely to be uniform. We find that, in contrast to the overall entropy bias, which is independent of the shape of the distribution (as long as it satisfies the right symmetry conditions), the probability $P(f)$ of obtaining function f within a set \mathbb{F}_t can depend on distribution shape. Nevertheless, for a given distribution shape, the probabilities $P(f)$ are independent of scale of the shape, e.g. they are independent of the variance of the Gaussian, or the width of the uniform distribution. This is because the function is invariant under scaling all weights by the same factor (true only in the case of no threshold bias). We will address the probabilities of functions within a given \mathbb{F}_t further in Section 3.3.3.

3.3.2 Simplicity bias of the $b = 0$ perceptron

The entropy bias of Theorem 3.3.1 entails an overall bias towards low Boolean complexity. In Theorem C.3.1 in Section C.3 we show that the Boolean complexity of a function f is bounded by¹

$$K_{\text{Bool}}(f) < 2 \times n \times \min(\mathcal{T}(f), 2^n - \mathcal{T}(f)). \quad (3.3)$$

¹A tighter bound is given in Theorem C.3.2, but this bound lacks any obvious closed form expression.

Using Theorem 3.3.1 and Equation (3.3), we have that the probability that a randomly initialised perceptron expresses a function f of Boolean complexity k or greater is upper bounded by

$$P(K_{\text{Bool}}(f) \geq k) < 1 - \frac{k \times 2^{-n} \times 2}{2 \times n} = 1 - \frac{k}{2^n \times n}. \quad (3.4)$$

Uniformly sampling functions would result in $P(K_{\text{Bool}}(f) \geq k) \approx 1 - 2^{k-2^n}$ which for intermediate k is much larger than Equation (3.4). Thus from entropy bias alone, we see that the perceptron is much more likely to produce simple functions than complex functions: it has an inductive bias towards simplicity. This derivation is complementary to the AIT arguments from *simplicity bias* [Dingle et al., 2018c, Valle-Pérez et al., 2018], and has the advantage that it also proves that bias exists, whereas AIT-based simplicity bias arguments presuppose bias.

To empirically study the inductive bias of the perceptron with $b = 0$, we sampled over many random initialisations with weights drawn from Gaussian or uniform distributions and input size $n = 7$. As can be seen in Figure 3.1a and Figure 3.1b, the probability $P(f)$ that function f obtains varies over many orders of magnitude. Moreover, there is a clear simplicity bias upper bound on this probability, which, as predicted by Eq. 2.1, decreases with increasing Lempel-Ziv complexity ($K_{LZ}(f)$) (using a version from [Dingle et al., 2018c] applied to the Boolean functions represented as strings of bits, see Section E.2). Similar behaviour was observed in [Valle-Pérez et al., 2018] for a FCN network. Moreover it was also shown there that Lempel-Ziv complexity for these Boolean functions correlates with approximations to the Boolean complexity K_{Bool} . A one-layer neural network (Figure 3.1c) shows stronger bias than the perceptron, which may be expected because the former has a much larger expressivity. A rough estimate of the slope a in Eq. 2.1 from [Dingle et al., 2018c] suggests that $a \sim \log_2(N_O)/\max_{f \in \mathbb{O}}(\tilde{K}(f))$ where \mathbb{O} is the set of all Boolean functions the model can produce, and N_O is the number of such functions. The maximum $K(f)$ may not differ that much between the one layer network and the perceptron, but N_O will be much larger in former than in the latter.

In Section E.0.1 we also show rank plots for the networks from Figure 1. Interestingly, at larger rank, they all show a Zipf like power-law decay, which can be used to estimate N_O , the total number of Boolean functions the network can express. We also note that the rank plots for the perceptron with $b = 0$ with Gaussian or uniform distributions of weights are nearly

indistinguishable, which may be because the overall rank plot is being mainly determined by the entropy bias.

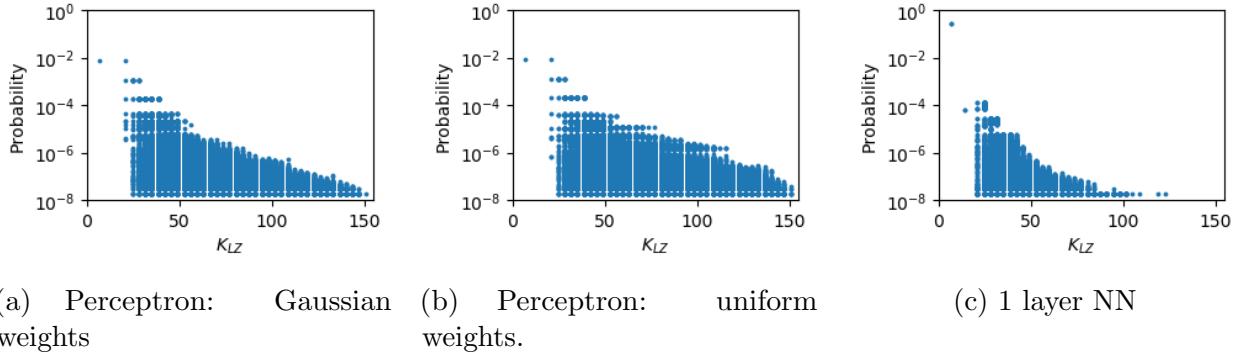


Figure 3.1: Probability $P(f)$ that a function obtains upon random choice of parameters versus Lempel-Ziv complexity $K_{LZ}(f)$ for (a) an $n = 7$ perceptron with $b = 0$ and weights sampled from a Gaussian distribution, (b) an $n = 7$ perceptron with $b = 0$ and weights sampled from a uniform distribution centred at 0 and (c) a 1-hidden layer neural network (with 64 neurons in the hidden layer). Weights w and the threshold bias terms are sampled from $\mathcal{N}(0, 1)$. For all cases 10^8 samples were taken and frequencies less than 2 were eliminated to reduce finite sampling effects. We present the graphs with the same scale for ease of comparison.

3.3.3 Bias within \mathbb{F}_t

In Figure 3.2 we compare a rank plot for all functions expressed by an $n = 7$ perceptron with $b = 0$ to the rank plots for functions with $\mathcal{T}(f) = 47$ and $\mathcal{T}(f) = 64$. To define the rank, we order the functions by decreasing probability, and then the rank of a function f is the index of f under this ordering (so the most probable function has rank 1, the second rank 2 and so on). The highest probability functions in \mathbb{F}_{64} have higher probability than the highest in \mathbb{F}_{47} because the former allows for simpler functions (such as 0101..), but for both sets, the maximum probability is still considerably lower than the maximum probability functions overall.

In Section E.2 we present further empirical data that suggests that these probabilities are bounded above by Lempel-Ziv complexity (in agreement with [Valle-Pérez et al., 2018]). However, in contrast to Theorem 3.3.1 which is independent of the parameter distribution (as long as they are symmetric), the distributions within \mathbb{F}_t are different for the Gaussian and uniform parameter distributions, with the latter showing less simplicity bias within a class of fixed t (see Section E.2.1).

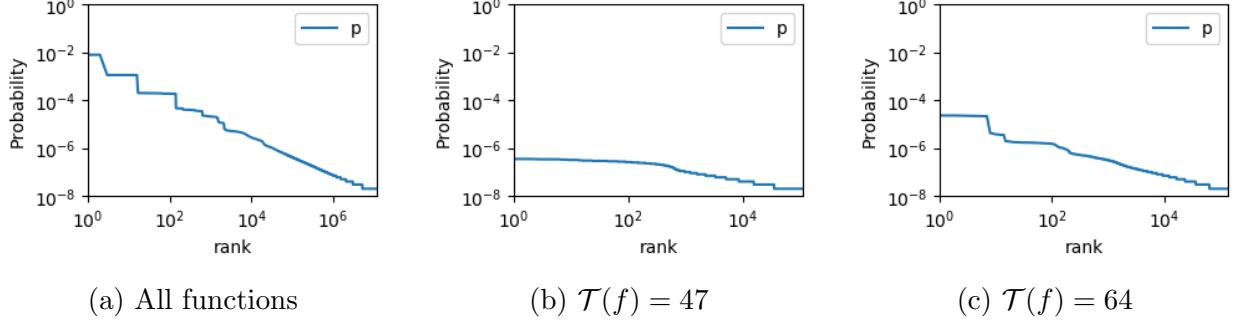


Figure 3.2: Probability $P(f)$ vs rank for functions for a perceptron with $n = 7$, $\sigma_b = 0$, and weights sampled from independent Gaussian distributions. In Figures 3.2b and 3.2c the functions are ranked within their respective \mathbb{F}_t . The seven highest probability functions in Figure 3.2c are $f = 0101\dots$ and equivalent functions obtained by permuting the input dimensions – note that these are very simple functions (simpler than the simplest functions that satisfy $\mathcal{T}(f) = 47$).

In Section E.3, we give further arguments for simplicity bias, based on the set of constraints that needs to be satisfied to specify a function. Every function f can be specified by a minimal set of linear conditions on the weight vector of the perceptron, which correspond to the boundaries of the cone in weight space producing f . The Kolmogorov complexity of conditions should be close to that of the functions they produce as they are related to the functions in a one-to-one fashion, via a simple procedure. In Section E.3, we focus on conditions which involve more than two weights, and show that within each set \mathbb{F}_t there exists one function with as few as 1 such conditions, and that there exists a function with as many as $n - 2$ such conditions. We also compute the set of necessary conditions (up to permutations of the axes) explicitly for functions with small t , and find that the range in the number and complexity of the conditions appears to grow with t , in agreement, with what we observe in Figure 3.2 for the range of complexities. More generally, we find that complex functions typically need more conditions than simple functions do. Intuitively, the more conditions needed to specify a function, the smaller the volume of parameters that can generate the function, so the lower its *a-priori* probability.

3.3.4 Effect of b (the threshold bias term) on $P(t)$

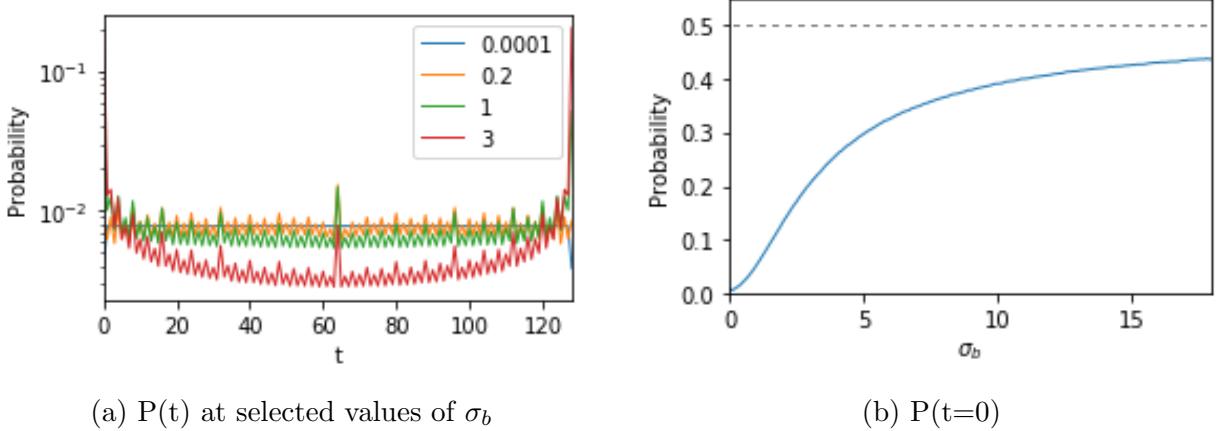


Figure 3.3: Effect of adding a bias term sampled from $\mathcal{N}(0, \sigma_b)$ to a perceptron with weights sampled from $\mathcal{N}(0, 1)$. (a) Increasing σ_b increases the bias against entropy, and with a particular strong bias towards $t = 0$ and $t = 2^n$. (b) $P(t = 0)$ increases with σ_b and asymptotes to $1/2$ in the limit $\sigma_b \rightarrow \infty$.

We next study the behaviour of the perceptron when we include the threshold bias term b , sampled from $\mathcal{N}(0, \sigma_b)$, while still initialising the weights from $\mathcal{N}(0, 1)$, as in Section 3.3.1. We present results for $n = 7$ in Figure 3.3. Interestingly, for infinitesimal σ_b , $P(T = 0)$ is less than for $b = 0$ (See Section C.2), but then for increasing σ_b it rapidly grows larger than $1/2^n$ and in the limit of large σ_b asymptotes to $1/2$ (see Figure 3.3b). It's not hard to see where this asymptotic behaviour comes from, a large positive or negative b means all inputs are mapped to true (1) or false (0) respectively.

We have focused our attention on the case where the inputs correspond to the corners of the uncentered hypercube $\{0, 1\}^n$. Here we show that the bias towards low entropy is recovered for the perceptron on centered ($\{-1, 1\}^n$) inputs, when the bias term is increased.

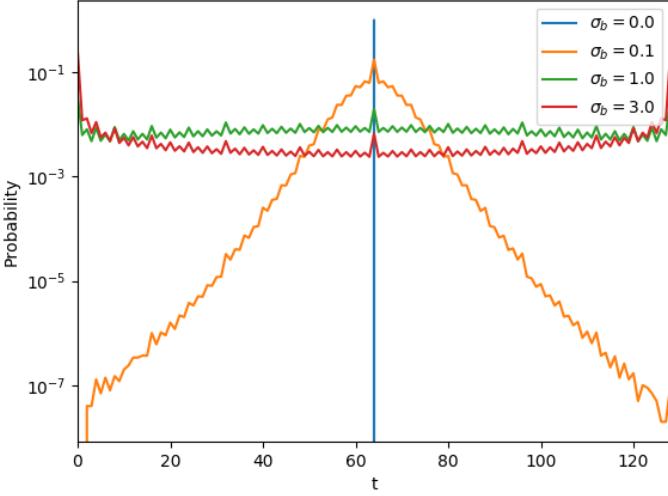


Figure 3.4: $\sigma_b = 1.0$

Figure 3.5: Probability of different values of T for the perceptron evaluated on $\{-1, 1\}^n$ inputs and varying $\sigma_b = 0.0$. The parameters were sampled 10^7 times. The weights were sampled i.i.d. from a Gaussian, with parameters $\sigma_w = 1.0$.

3.4 Entropy bias in Multi-layer Neural Networks

We next extend results from Section 3.3 to multi-layer neural networks, with the aim to comment on the behaviour of $P(T = t)$ as we add hidden layers with ReLU activations.

To study the bias in the parameter-function map of neural networks, it is important to first understand the expressivity of the networks. In Section 3.4.1, we produce a (loose) upper bound on the minimum size of a network with ReLU activations and l layers that is maximally expressive over Boolean functions. We comment on how sufficiently large expressivity implies a larger bias towards low entropy for models with similarly shaped distribution over T (when compared to the perceptron).

In Section 3.4.2, we prove, in the limit of infinite width, that adding ReLU activated layers causes the moments of $P(T = t)$ to increase, . This entails a lower expected entropy for neural networks with more hidden layers. We empirically observe that the distribution of T becomes convex (with input $\{0, 1\}^n$) with the addition of ReLU activated layers for neural networks with finite width.

3.4.1 Expressivity conditions for DNNs

We provide upper bounds on the minimum size of a DNNs that can model all Boolean functions. We use the notation $\langle n_0, n_1, \dots, n_L, n_{L+1} \rangle$ to denote a neural network with ReLU activations and of the form given in Theorem 2.1.1.

Lemma 3.4.1. *A neural network with layer sizes $\langle n, 2^{n-1}, 1 \rangle$, threshold bias terms, and ReLU activations can express all Boolean functions over n variables (also found in [Raj, 2018]). See Mingard et al. [2019] for proof.*

Lemma 3.4.2. *A neural network with l hidden layers, layer sizes $\langle n, (n + 2^{n-1}/l + 1), \dots, (n + 2^{n-1}/l + 1), 1 \rangle$, threshold bias terms, and ReLU activations can express all Boolean functions over n variables. See Mingard et al. [2019] for proof.*

Note that neither of these bounds are (known to be) tight. Theorem 3.4.1 says that a network with one hidden layer of size 2^{n-1} can express all Boolean functions over n variables. We know that a perceptron with n input neurons (and a threshold bias term) can express at most 2^{n^2} Boolean functions ([Anthony, 2001], Theorem 4.3), which is significantly less than the total number of Boolean functions over n variables, which is 2^{2^n} . Hence there is a very large number of Boolean functions that the network with a (sufficiently wide) hidden layer can express, but the perceptron cannot. The vast majority of these functions have high entropy (as almost all Boolean functions do). Moreover, we observe that the measure $P(T = t)$ is convex in the case of the more expressive neural networks, as discussed in section Section 3.4.2. This suggests that the networks with hidden layers have a much stronger relative bias towards low entropy functions than the perceptron does, which is also consistent with the stronger simplicity bias found in Figure 3.1.

We further observe from Theorem 3.4.2 that the number of neurons can be kept constant and spread over multiple layers without loss of expressivity for a Boolean classifier (provided the neurons are evenly spread across the layers).

3.4.2 How multiple layers affect the bias

We next consider the effect of addition of ReLU activated layers on the distribution $P(t)$. Of course adding even just one layer hugely increases expressivity over a perceptron. Therefore,

even if the distribution of $P(t)$ would not change, the average probability of functions in a given \mathbb{F}_t could drop significantly due to the increase in expressivity.

However, we observe that for inputs $\{0, 1\}^n$, $P(t)$ becomes more convex when more ReLU-activated hidden layers are added, see Figure 3.6. The distribution appears to be monotone on either side of $t = 2^{n-1}$ and relatively flat in the middle, even with the addition of 8 intermediate layers². In particular, we show in Figure 3.6 that for large number of layers, or large σ_b , the probabilities for $P(t = 0)$ (and by symmetry, in the infinite width limit, also $P(t = 2^n)$) each asymptotically reach $\frac{1}{2}$, and thus take up the vast majority of the probability weight.

We now prove some properties of the distribution $P(t)$ for DNNs with several layers.

Lemma 3.4.3. *The probability distribution on T for inputs in $\{0, 1\}^n$ of a neural network with linear activations and i.i.d. initialisation of the weights is independent of the number of layers and the layer widths, and is equal to the distribution of a perceptron. See Section C.4 for proof.*

While it is trivial that such a linear network has the same expressivity as a perceptron, it may not be obvious that the entropy bias is identical.

Lemma 3.4.4. *Applying a ReLU function in between each layer produces a lower bound on $P(T = 0)$ such that $P(T = 0) \geq 2^{-n}$. See Section C.4 for proof.*

This lemma shows that a DNN with ReLU functions is no less biased towards the lowest entropy function than a perceptron is. We prove a more general result in the following theorem which concerns the behaviour of the average entropy $\langle H(t) \rangle$ (where the average upon random sampling of parameters) as the number of layers grows. The theorem shows that the bias towards low entropy becomes stronger as we increase the number of layers, for any distribution of inputs. We rely on previous work that shows that in the infinite width limit, neural networks approach a Gaussian process (Lee et al. [2018], Garriga-Alonso et al. [2018], Novak et al. [2018a], Matthews et al. [2018], Yang [2019a]), which for the case of fully-connected ReLU networks, has an analytic form [Lee et al., 2018].

Theorem 3.4.5. *Let \mathbb{S} be a set of $m = |\mathbb{S}|$ input points in \mathbb{R}^n . Consider neural networks with i.i.d. Gaussian weights with variances σ_w^2/\sqrt{n} and biases with variance σ_b , in the limit where*

²Note that this is when the input is $\{0, 1\}^n$. This is not true for all input distributions. See, e.g. Figure 3.6d. The change in probability for other distributions can be more complex.

the width of all hidden layers n goes to infinity. Let N_1 and N_2 be such neural networks with L and $L + 1$ infinitely wide hidden layers, respectively, and no bias. Then, the following holds: $\langle H(T) \rangle$ is smaller than or equal for N_2 than for N_1 . It is strictly smaller if there exist pairs of points in \mathbb{S} with correlations less than 1. If the networks have sufficiently large threshold bias ($\sigma_b > 1$ is a sufficient condition), the result above also holds. For smaller bias, the result holds only for a sufficiently large number of layers.

See Section C.4 for a proof of Theorem 3.4.5. Theorem 3.4.5 is complementary to Theorem 3.3.1, in that the former only proves that the bias towards low entropy increases with depth, and the later proves conditions on the data that guarantee bias toward low entropy on the “base case” of 0 layers. We show in Figure 3.6 that when $\sigma_b = 0$, the bias towards low entropy indeed becomes monotonically stronger as we increase the number of ReLU layers, for both inputs in $\{0, 1\}^n$ as well as for centered data $\{-1, 1\}^n$.

For centered inputs $\{-1, 1\}^n$, the perceptron with $b = 0$ shows rather unusual behaviour. The distribution is completely peaked around $t = 2^{n-1}$ because every input mapping to 1 has the opposite input mapping to 0. Not surprisingly, its expressivity is much lower than the equivalent perceptron with $\{0, 1\}^n$ (as can be seen in Figure E.2a in Section E.0.1). Nevertheless, in Figure 3.6d we see that as the number of layers increases, the behaviour rapidly resembles that of uncentered data (In Section 3.6 we also show that the bias toward low entropy is also recovered as we increase σ_b). So far this is the only exception we have found to the general bias to low entropy we observe for all other systems (see also Section 3.5). We therefore argue that this is a singular result brought about by particular symmetries of the perceptron with zero bias term. The fact that there is an exception does not negate our general result which we find holds much more generally.

The insets of in Figure 3.6 show that the two trivial functions asymptotically dominate in the limit of large numbers of layers. We note that recent work ([Lee et al., 2018, Luther and Seung, 2019]) has also pointed out that for fully-connected ReLU networks in the infinite-width infinite-depth limit, all inputs become asymptotically correlated, so that the networks will tend to compute the constant function. Here we give a quantitative characterisation of this phenomenon for any number of layers.

Some interesting recent work [Yang and Salman, 2019] has shown that certain choices of

hyperparameters lead to networks which are a priori unbiased, that is the $P(f)$ appears to be uniform. In Section E.1 we show that this result is due to a choice of hyperparameters that lie deep in the chaotic region defined in [Poole et al., 2016]. The effect therefore depends on the choice of activation function (it can occur for say tanh and erf, but most likely not ReLU), and we are studying it further.

3.5 Bias towards low entropy in realistic datasets and architectures

Up to now we have demonstrates bias towards low entropy only for the Perceptron, where we can prove certain results rigorously, and for a fully connected network (FCN). An obvious question is: does this bias persist for other architectures or data sets? In this section we show that similar bias indeed occurs for more realistic datasets and architectures.

We perform experiments whereby we sample the parameters of a convolutional neural network (CNN) (with a single Boolean output), and evaluate the functions they obtain on subsets of the standard vision datasets MNIST or CIFAR10. Our results suggest that the bias towards low entropy (high class imbalance) is a generic property of ReLU-activated neural networks. This has in fact been pointed out previously, either empiricallyPage [2019], or for the limit of infinite depthLee et al. [2018]. Extending our analytic results to these more complicated architectures and datasets is probably extremely challenging, but perhaps possible for some architectures, by analyzing the infinite-width Gaussian process (GP) limit.

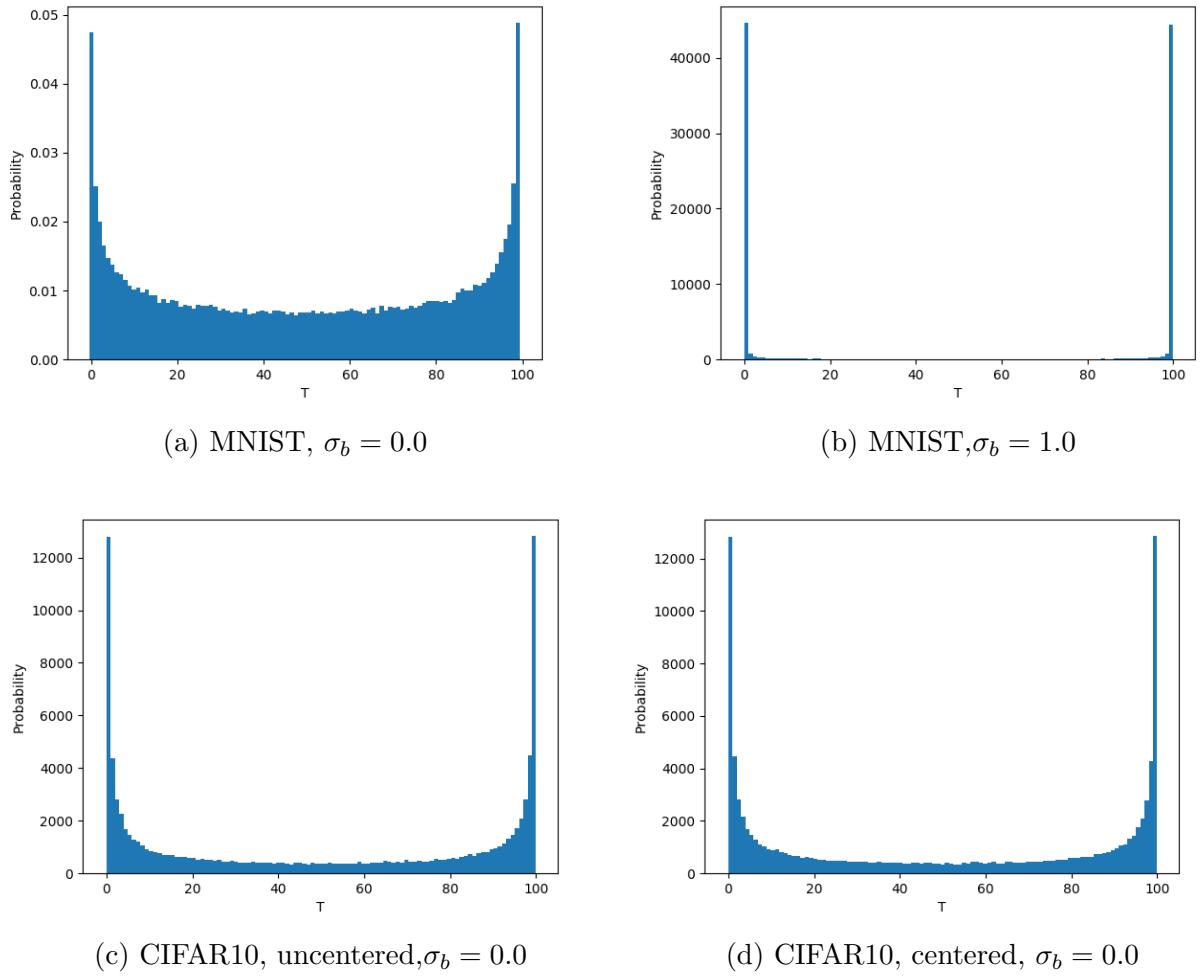


Figure 3.7: Probability of different values of T for a CNN with 4 layers, no pooling, and ReLU activations. The parameters were sampled 10^5 times and the network was evaluated on a fixed random sample of 100 images from MNIST or CIFAR10. The parameters were sampled i.i.d. from a Gaussian, with parameters $\sigma_w = 1.0$, and $\sigma_b = 0.0, 1.0$. The input images from CIFAR10 where either left uncentered (with values in range $[0, 1]$) (*uncentered*), or where centered by subtracting the mean value of every pixel (*centered*).

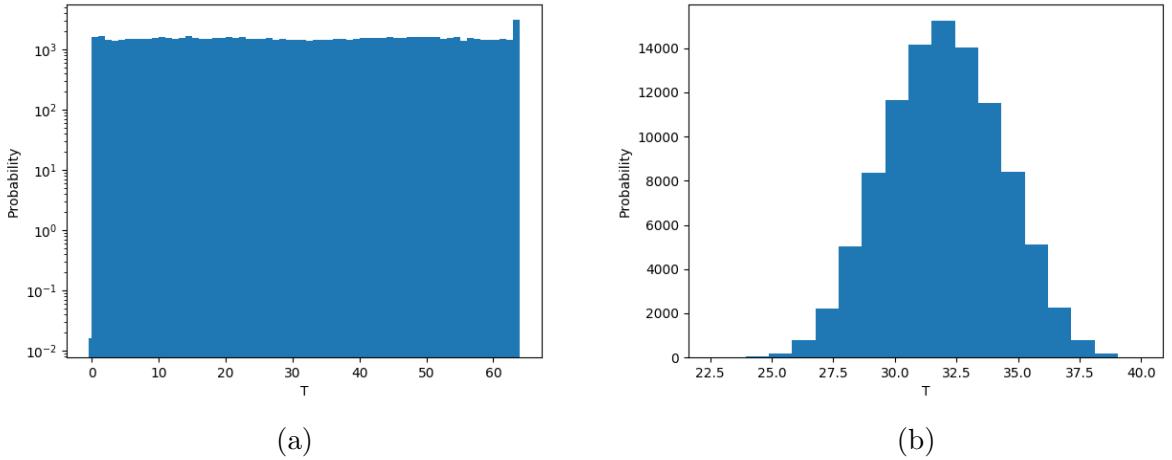


Figure 3.8: Probability of different values of T for the perceptron with $n = 7$ input neurons and $\sigma_b = 0.0$. The parameters were sampled 10^5 times and the perceptron was evaluated on a random subsample of size 64 of either $\{0, 1\}^n$ ((a)) or $\{-1, 1\}^n$ ((b)). The weights were sampled i.i.d. from a Gaussian, with parameters $\sigma_w = 1.0$.

3.6 Effect of the bias term on the perceptron on centered data

Here we show that the bias towards low entropy is recovered for the perceptron on centered ($\{-1, 1\}^n$) inputs, when the bias term is increased.

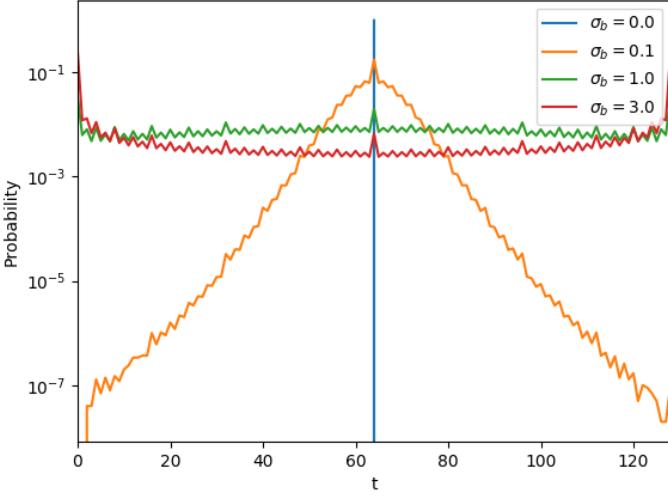


Figure 3.9: $\sigma_b = 1.0$

Figure 3.10: Probability of different values of T for the perceptron evaluated on $\{-1, 1\}^n$ inputs and varying $\sigma_b = 0.0$. The parameters were sampled 10^7 times. The weights were sampled i.i.d. from a Gaussian, with parameters $\sigma_w = 1.0$.

3.7 Effect of bias on learning

The effect of inductive biases such as the ones we discuss on learning is a vast and open research project. In particular, we expect that bias towards low entropy should play a bigger role when trying to learn class-imbalanced data. In class-imbalanced problems, one is often interested in quantities beyond the raw test accuracy. For example, the *sensitivity* (the fraction of test-set misclassifications on the rare class), and the *specificity* (the fraction of test-set misclassifications on the common class). Furthermore, the use of tricks, like oversampling the rare class during optimisation, makes the formal analysis of practical cases even more challenging. In this section, we show preliminary results illustrating some of the effects that entropy bias in $P(f)$ can have on learning.

In the experiments, we shift the bias term of the output neuron $b \mapsto b' = b + \text{shift}$. This causes the functions at initialisation to have distributions $P(T)$ which may be biased towards higher or lower values of T (mapping most inputs to 0 or 1). We measure the average $\langle T \rangle$ for different choices of the shift hyperparameter, and train the network, using SGD, on a

class-imbalanced dataset. Here we perform this experiment on a CNN with 4 layers (and no pooling), and a FCN with 1 layer. The dataset consists of a sample of either MNIST (10 classes) or balanced EMNIST (47 classes), where a single class is labelled as 0, and the rest are labelled as 1.

We see in Figure 3.11 that values of the shift hyperparameter producing large values $\langle T \rangle$ give higher test accuracy. We suggest that this could be because the new parameterisation induced by the shifted bias term³, causes the inductive bias is more “attuned” to the correct target function (which has a class imbalance of 1:10 and 1:47, respectively for MNIST and balanced EMNIST). However, the detailed shape of the curve seems architecture and data dependent. For instance, for the FCN (with ReLU or tanh activation) trained on MNIST we find a clear accuracy peak around the true value of $\langle T \rangle$, but not for the others.

We also measured the sensitivity and 99% specificity (found by finding the smallest threshold value of the sigmoid unit, giving 99% specificity). Sensitivity seems to follow a less clear pattern. For two of the experiments, it followed the accuracy quite closely, while for FCN trained on MNIST it peaked around a positive value of the shift hyperparameter.

These results suggest that looking at architectural changes that affect the entropy bias of $P(f)$ can have a significant effect on learning performance for class-imbalanced tasks. For example, for the tanh-activated FCN, increasing the shift hyperparameter above 0 improves both the accuracy and sensitivity significantly. As we mentioned at the beginning, understanding the full effect of entropy bias, and other biases on $P(f)$, on learning is a large research project. In this chapter, we focused on explaining properties of $P(f)$, while only showing preliminary results on the effect of these properties on learning. In Chapter 5 and Chapter 6 we will tackle this question more in depth.

³Note that the shifted bias reparameterisation is equivalent to a shifted *initialization* in the original parameterisation. For more complex reparameterisation, this may not be the case

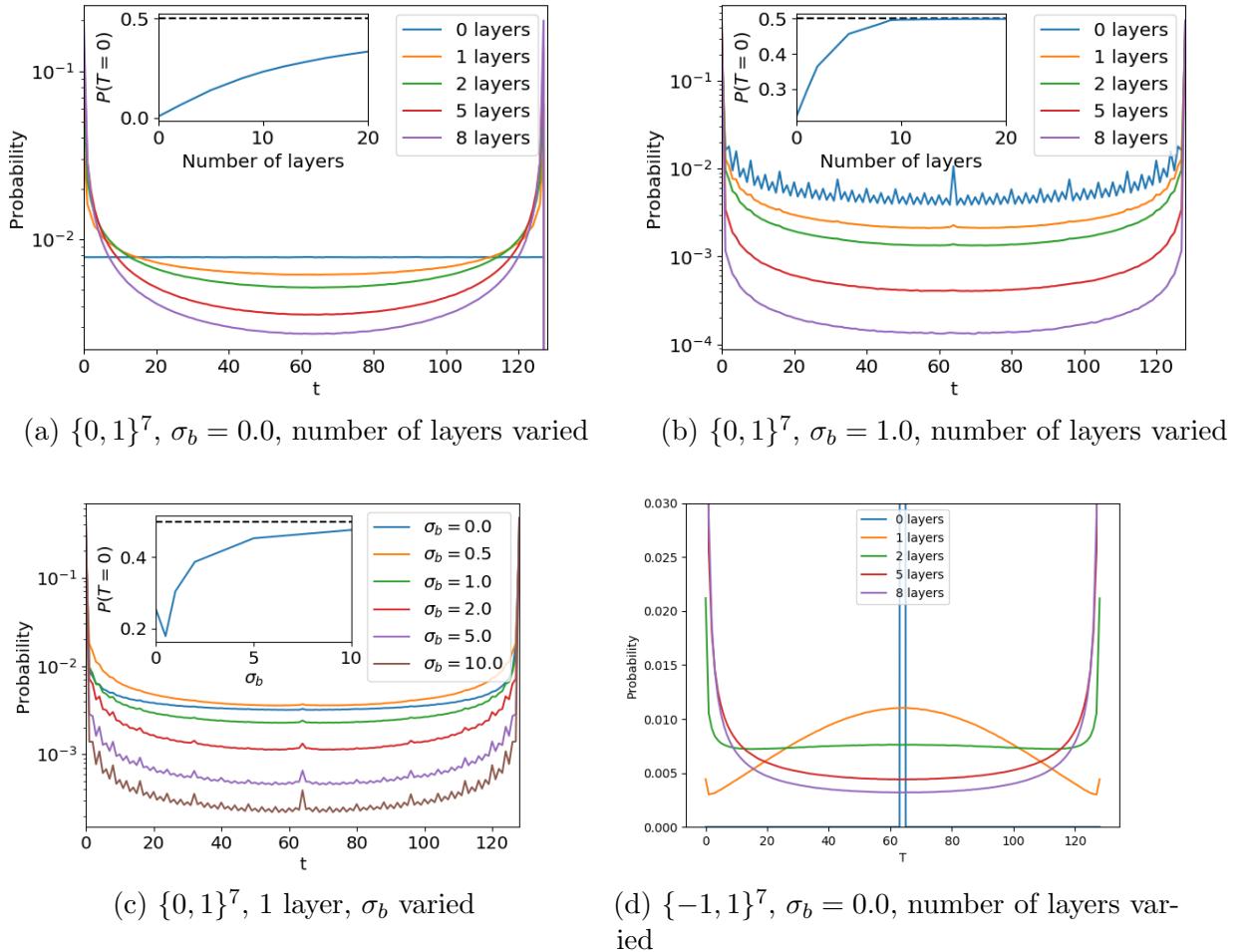


Figure 3.6: **$P(T = t)$ becomes on average more biased towards low entropy for increasing number of layers or increasing σ_b .** Here we use $n = 7$ input layers, with input $\{0, 1\}^7$ (*centered data*) or $\{-1, 1\}^7$ (*uncentered data*). The hidden layers are of width $2^{n-1} = 64$ to guarantee full expressivity. $\sigma_w = 1.0$ in all cases. The insets show how $P(t = 0)$ asymptotes to $\frac{1}{2}$ with increasing layers or σ_b .

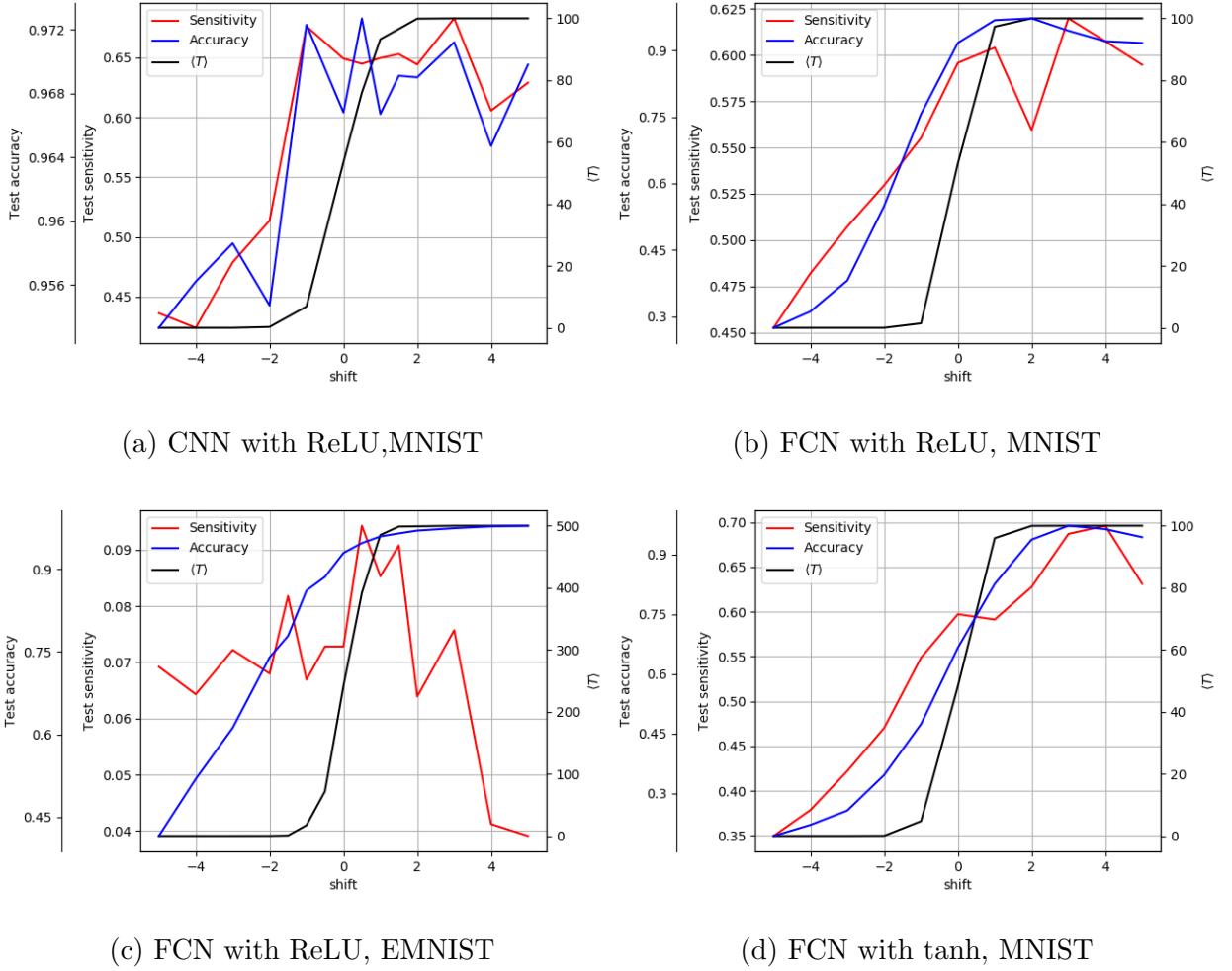


Figure 3.11: Test accuracy, sensitivity (at 99% specificity), and $\langle T \rangle$ versus the shift hyper-parameter (shift of the bias term in the last layer), for different architectures and datasets. The networks were trained with SGD (batch size 32, cross entropy loss), until reaching 100% training accuracy. Accuracies and sensitivities are averages over 10 SGD runs with random initializations ($\sigma_b = 0.0$, $\sigma_w = 1.0$). The dataset had size 100 for MNIST, and 500 for EMNIST. Images are centered, so pixel values have their mean over the whole dataset subtracted. The values of $\langle T \rangle$ are estimated from 100 random parameter samples and evaluating on the same data we train the network on.

Chapter 4

Generalization theory of deep learning

We now revisit one of the central questions we are tackling in this thesis: ‘Why don’t heavily parameterized neural networks overfit the data?’. This is a question about generalization – the ability for a learning algorithm to predict correctly in data not observed during training. As we will see statistical learning theory has developed many tools to understand generalization, and many of these have been applied to DNNs. In Chapter 2 we proposed a novel perspective on generalization in DNNs based on the simplicity bias of their parameter function map. Learning theory will give us the tools to make that intuition quantitative.

Already in 1996, Bartlett sharpened the question of why overparametrized neural networks generalize, by pointing out that standard approaches to learning theory were inadequate: “the VC-bounds seem loose; neural networks often perform successfully with training sets that are considerably smaller than the number of network parameters”, and proposed an alternative approach based on the norm of the weights of the DNN [Bartlett, 1996]. Lawrence et al. [1996] proposed that backpropagation could be responsible for the implicit bias towards low norm solutions, in part because this can be shown for some analytically tractable linear models. We will see in this chapter that linking generalization to low weight norm (for some notion of norm) is one of the ideas that have been recently applied to understand deep learning, although with limited success.

More recently, Zhang et al. [2017a] revived these same concerns in the light of the recent successes in dep learning. They observed that “classical” approaches in learning theory, based on worst-case analyses, are not sufficient to explain generalization of deep learning models,

and also proposed that SGD could be biasing towards small norm solutions. The observations in Zhang et al. [2017a] have spurred a lot of work on applying and developing other learning theory approaches to deep learning, some of which are showing promising results. However, current approaches are still lacking in many ways: generalization error bounds are often vacuous or offer little explanatory power by showing opposite trends to the true error as one varies important quantities. For example, Nagarajan and Kolter [2019] show that some common spectral norm-based bound predictions increase with increasing training set size, while the measured error decreases.

There is therefore a need for a more systematic approach to studying generalisation bounds. Two important recent studies have taken up this challenge. Firstly, Jiang et al. [2019] preformed extensive empirical tests of the predictive power of over 40 different measures of generalization for a range of commonly used hyperparameters. They mainly studied flatness and norm-based measures, and compared them by reporting an average of the performance. They focused mostly on training hyperparameters, with only width and depth being varied for the architecture, and did not systematically study variation on the data (dataset complexity or training set size).

Secondly, in a follow-up study Dziugaite et al. [2020], argue that average performance is not a sufficient metric of generalisation performance. They demonstrate that generalisation measures can exhibit qualitatively correct correlations for certain parameter changes and seeds, but badly fail for others. In addition to some of the hyperparameter changes studied by Jiang et al. [2019], they also consider the effect of changing dataset complexity and training set size.

In this chapter and the next we also perform a systematic analysis of generalisation bounds. We first describe, in Section 4.1, a set of 7 qualitatively different desiderata which, we argue, should be used to guide the kind of experiments¹ that are used to test the predictions of a theory of the generalization. These include 4 desiderata for making correct qualitative predictions when varying architecture, dataset complexity, dataset size, or optimizer choice, as well as 3 slightly more general desiderata, which describe the importance of quantitative agreement, computational efficiency and theoretical rigour.

Rather than empirically testing the many bounds that can be found in the literature, we take a more general theoretical approach here. We first, in Section 4.2, present an overall

¹Here, ‘experiment’ can be interpreted in the formal sense which Dziugaite et al. [2020] introduce

framework for classifying the many different approaches to deriving frequentist bounds of the generalization error. We use this framework to organize the discussion of the different approaches. For each approach we review existing results in the literature in light of the 7 desiderata we propose. Such an analysis allows us to draw some more general conclusions about what kinds of strategies for deriving generalisation bounds may be most successful. In this context we also note, that for some proposed bounds, there is not enough empirical evidence in the literature to decide whether or not they satisfy our desiderata. This lacuna highlights the importance of the proposal put forward by Jiang et al. [2019] and Dziugaite et al. [2020] of large scale empirical studies to understand generalization, which we also advocate for in this paper.

Inspired by the overall analysis we present here of existing bounds, we next derive and extensively test a new realizable high-probability PAC-Bayes bound based on the original PAC-Bayes bound by McAllester [McAllester, 1998]. Because our bound, which is derived from a function-based picture, is directly proportional to the Bayesian evidence or marginal likelihood of the model, we call it the marginal likelihood PAC-Bayes bound. While our bound has frequentist origins, the connection between marginal likelihood and generalization is an important theme in Bayesian analyses, and can be traced back to the work of MacKay and Neal [MacKay, 1992a, Neal, 1994, MacKay and Mac Kay, 2003], as well as the early work on PAC-Bayes [Shawe-Taylor and Williamson, 1997, McAllester, 1998]. See also an important recent discussion of this connection from a Bayesian perspective in Wilson and Izmailov [2020].

We describe our marginal-likelihood PAC-Bayes bound in more detail in Section 4.4. In particular, we prove that it is asymptotically optimal up to a constant in the limit of large training sets, upon the assumption that the learning curve has a power law asymptotic behaviour with training set size.

In our discussion in Chapter 7, we argue that a key to understanding the relatively good performance of our bound across so many desiderata is our use of a function based perspective, in contrast to many other PAC-Bayesian bounds in the deep learning theory literature, which use distributions in parameter space, mostly as a way to measure flatness. We also explore in Chapter 6 why our theory, which is rigorously proven only for Bayesian DNNs, predicts well the behaviour of SGD-trained DNNs.

More generally, we argue that capturing trends in generalization when architecture and data are varied is not only theoretically interesting, but also important for many applications including architecture search and data augmentation.

4.1 Desiderata for predictive theories of generalization error

The fundamental question of what constitutes a “good” theory of generalisation in deep learning is a profound and contested one in the literature. And if one proposed such a theory, then a full causal analysis of its performance should involve a careful and thorough analysis of all the relevant variables in the learning problem and associated algorithms. In principle such an analysis can be done with simplifying assumptions. For example, in Jiang et al. [2019] a number of generalisation bounds are compared for a limited number of variations, and the average performance of each bound is reported. In Dziugaite et al. [2020] a similar notion to define a good predictive theory of generalization is suggested. They furthermore formalize this notion using distributional robustness, which quantifies the performance of the theory in the worst case (instead of the average case measure of Jiang et al. [2019]).

Here we take a slightly different approach to this question by defining a series of seven key desiderata that a ”good” theory of generalisation should satisfy. This set is larger than the number experimental combinations used in Jiang et al. [2019], Dziugaite et al. [2020], and so in that sense it is a more demanding ask. Ideally a theory would make predictions that correlate with experimental observation over as many desiderata as possible. However, rather than defining an average or a worst-case measure, we take a more pragmatic approach, and intentionally treat the concept of “good predictive theory of generalization” in a more informal fashion, because depending on the application, different desiderata may be more relevant.

We focus our attention on generalization error upper bounds, but the same ideas could apply to other types of theories that aim to predict generalization error (for example, those based on Bayesian assumptions on the data distribution).

The seven desiderata which we propose a good predictive theory of generalization for deep learning should aim to satisfy are listed below:

- **D.1 - data complexity:** The predicted error should correctly scale with data complexity. In other words, the predicted error should correlate well with the true error when the dataset is changed. For example, a fixed DNN, for a fixed training set size, may have higher error for CIFAR10 than for MNIST, and an even higher error for a label-corrupted dataset. The bound should capture these differences.
- **D.2 - training set size:** The predicted error should correctly scale with training set size. That is, the predicted error should follow the same trend as the true error as the number of training examples increases. It has been found empirically that the generalization error often follows a power law decay with training set size with an exponent which depends on the dataset, but not strongly on the architecture [Hestness et al., 2017, Novak et al., 2019b, Rosenfeld et al., 2019, Kaplan et al., 2020].
- **D.3 - architectures:** The predicted error should capture differences in generalization between architectures. Different architectures can display significant variation in generalization performance. For example, CNNs with pooling tend to generalize better than CNNs without pooling on image classification tasks. The predicted error should aim to predict these differences. Furthermore, one of the most puzzling properties of DNNs is that the performance depends only weakly on the number of parameters used for an architecture, provided the system is large enough Belkin et al. [2019], Nakkiran et al. [2019]. Since this question of why DNNs generalize so well in the overparameterised regime is one of the crucial questions in the theory of DNNs, it is particularly important that a predicted error can reproduce this insensitivity to the number of parameters.
- **D.4 - optimization algorithms:** The predicted error should capture differences in generalization between different optimization algorithms. Different optimization algorithms used in training DNNs, as well as different choices of training hyperparameters such as SGD batch size and learning rates, or different regularization techniques, can lead to differences in generalization, which the theory should aim to predict.

- **D.5 - non-vacuous:** The predicted error should be quantitatively close to the true error. For generalization error upper bounds, it is commonly required that the error is less than 100%, as any upper bound higher than that is satisfied by definition and thus can be considered “vacuous”. Beyond this requirement, the bound should aim to be as close as possible to the true error (a property often referred to as being a *tight* bound).
- **D.6 -efficiently computable:** The predicted error should be efficiently computable. This requirement is particularly important if the prediction or bound is to be useful in practical applications (for example, in architecture search). It is of little use having a bound that cannot in practice be calculated.
- **D.7 - rigorous:** The prediction should be rigorous. For the case of upper bounds, this means that the bound comes with a theorem that guarantees its validity under a well-specified set of assumptions, which should be met on the domain where it is applied. In practice, theoretical results are often applied to domains where the assumptions aren’t known to hold (or are known *not* to hold), but rigorous guarantees offer the highest level of confidence on one’s predictions, and should be aimed for whenever possible.

A theory of generalization error should aim to satisfy as many of these desiderata as possible. However, the trade-offs that one is willing to make depend on the domain of application of the theory. For example, a rigorous proof may be more valuable for theoretical insight than for a practical application, where computational efficiency may be more important. As another example, an application to neural architecture search may care more about the scaling with architecture, while an application to decision making for data collection may care more about correct scaling with training set size.

4.2 Classifying generalization bounds for deep learning

In this section, we provide an overview of different existing approaches to predict generalization error. After some preliminary sections describing notation and definitions we present a general formalism which classifies the different types of generalization error upper bounds based on key assumptions they make.

4.2.1 Notation for the supervised learning problem

Here we describe our notation using the standard formalization of the supervised learning problem. Supervised learning deals with the problem of predicting outputs from inputs, given a set of example input-output pairs. The inputs live in an input domain \mathcal{X} , and the outputs belong to an output space \mathcal{Y} . We will mostly focus on the binary classification setting where $\mathcal{Y} = \{0, 1\}$. We assume that there is a *data distribution* \mathcal{D} on the set of input-output pairs $\mathcal{X} \times \mathcal{Y}$. The *training set* S is a sample of m input-output pairs sampled i.i.d. from \mathcal{D} , $S = \{(x_i, y_i)\}_{i=1}^m \sim \mathcal{D}^m$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. We will refer to the data distribution as *noiseless* if it can be factored as $\mathcal{D}((x, y)) = \mathcal{D}_X(x)\mathcal{D}_{Y|X}(y|x)$ where $\mathcal{D}_{Y|X}$ is deterministic, that is it has support on a single value in $y = f(x)$. In this case f is called the *target function*.

We define a *loss function* $L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$, which measures how “well” a prediction $\hat{y} \in \mathcal{Y}$ matches an observed output $y \in \mathcal{Y}$, by assigning to it a score $L(\hat{y}, y)$ which is high when they don’t match. We define a *predictor* as a function from inputs to outputs $h : \mathcal{X} \rightarrow \mathcal{Y}$ and the *risk* R of a predictor as the expected value of the loss of the predicted outputs on new samples $R(h) = \mathbf{E}_{(x,y) \sim \mathcal{D}}[L(h(x), y)]$. For the case of classification, where \mathcal{Y} is a discrete set, we focus on the *0-1 loss* function (also called *classification loss*) $L(\hat{y}, y) = \mathbb{1}[\hat{y} \neq y]$, where $\mathbb{1}$ is the indicator function. We define the *generalization error* ϵ as the expected risk using this loss, which equals the probability of misclassification.

$$\epsilon(h) = \mathbf{P}_{(x,y) \sim \mathcal{D}}[h(x) \neq y] \quad (4.1)$$

This is the central quantity which we study here. We also define the *empirical risk* as the empirical average of the loss $\widehat{R}(h, S) = \frac{1}{m} \sum_{(x,y) \in S} L(h(x), y)$, and for classification we define the *training error* to be $\widehat{\epsilon}(h, S) = \frac{1}{m} \sum_{(x,y) \in S} \mathbb{1}[h(x) \neq y]$. For simplicity of notation we often simply write $\widehat{R}(h)$ and $\widehat{\epsilon}(h)$ for $\widehat{R}(h, S)$ and $\widehat{\epsilon}(h, S)$, respectively.

Finally, we define a *learning algorithm* to be a mapping $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^* \rightarrow \mathcal{Y}^\mathcal{X}$ from training sets of any size to predictor functions. For simplicity, we described deterministic predictors and learning algorithms, but most results should be easily generalizable to stochastic versions of these. A stochastic learning algorithm will map training sets to a probability distribution over predictors, while a stochastic predictor maps inputs in \mathcal{X} to probability distributions over

outputs in \mathcal{Y} . For PAC-Bayes we will consider stochastic learning algorithms.

The supervised learning problem consists of finding a learning algorithm that produces low generalization error (in expectation, or with high probability), under either no or some assumptions on the data distribution \mathcal{D} .

4.2.2 General PAC learning framework

The modern theory of statistical learning theory originated with Leslie Valiant's probably approximately correct (PAC) framework [Valiant, 1984], and Vapnik and Chervonenkis' uniform convergence analysis [Vapnik, 1968, Vapnik and Chervonenkis, 1974, Vapnik, 1995]. Here we introduce a general frequentist formulation of generalization error bounds of the type studied in supervised learning theory. We will define the main classes of bounds depending on the quantity bounded, and on whether realizability is assumed or not.

Although the original PAC framework included the condition of computational efficiency of the learning algorithm in its definition, the term is used more widely now [Guedj, 2019]. In the most general case, the PAC learning framework we present here proves confidence bounds on a function ϕ of the *observed* training set S , and the *unobserved* generalization error of the predictor output by the algorithm. That is, it proves statements of the following form: For a range of $\delta \in [0, 1]$, the following holds

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m}[\phi(\epsilon(\mathcal{A}(S)), S) \leq 0] \geq 1 - \delta \quad (4.2)$$

for a given ϕ , and under either no or some assumptions on the data distribution \mathcal{D} , and algorithm \mathcal{A} . We omit dependence of bounds on δ throughout the chapter in order to simplify notation. The idea is that we want to prove a certain property relating observed quantities and the unobserved generalization error, which is the real quantity of interest. Equation (4.2) is a very general frequentist bound. In the context of learning theory, ϕ almost always is expressed to take the following form

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m}[D(\epsilon(\mathcal{A}(S))), \hat{\epsilon}(\mathcal{A}(S)) \leq \frac{f_{\mathcal{A}}(S)}{m^\alpha}] \geq 1 - \delta \quad (4.3)$$

where $0 < \alpha \leq 1$, D is a function measuring the difference between generalization and training error (which we refer to as the *generalization gap*), and $f_{\mathcal{A}}$ is a function which we call *capacity* measuring some notion of “complexity” of the algorithm, the data, or both. The most common measure of generalization gap D is the absolute difference between the generalization and training error, but in some general PAC-Bayesian analyses D may be any convex function [Rivasplata et al., 2020]. The capacity $f_{\mathcal{A}}$ can take many forms, but some common examples include the VC dimension (section 4.3.1.1) and the KL divergence of a posterior and prior for PAC-Bayes bounds (section 4.3.2.1). Sometimes the dependence on training set is fully absorbed into $f_{\mathcal{A}}$ so that $1/m^{\alpha}$ is omitted. We also distinguish two types of bounds

- *Generalization gap bounds* are bounds on $|\epsilon(h) - \hat{\epsilon}(h)|$ or some other measure of discrepancy D between the generalization and training error
- *Generalization error bounds* are bounds on $\epsilon(h)$

Note that a generalization gap bound immediately implies a generalization error bound, but not necessarily vice versa. If $\hat{\epsilon}(h) = 0$ (realizability assumption), the distinction between these bounds doesn’t exist.

4.2.3 A formal classification of generalization bounds

We will now classify the main types of bounds which are possible under the general PAC framework described above, according to the different assumptions they make on the data distribution or algorithm, and on the different quantities the capacity $f_{\mathcal{A}}(S)$ may depend on.

4.2.3.1 According to assumptions on \mathcal{D}

The PAC framework is characterized by making bounds which make as little assumptions about the data distribution \mathcal{D} as possible. There are two main approaches: agnostic and realizable bounds.

Agnostic bounds. For *agnostic* or *distribution-free* bounds, no assumption is made on the data distribution. In this case, the No Free Lunch theorem [Wolpert and Waters, 1994] implies that we can’t guarantee a small generalization error, but may still be able to guarantee

a small difference between the training and generalization error (generalization gap), or a small generalization error for some training sets (for data-dependent bounds).

Realizable bounds. For *realizable* bounds, the data distribution and the algorithm are assumed to be such that $\hat{\epsilon}(\mathcal{A}(S)) = 0$ for any S which has non-zero probability. This is saying that the algorithm is always able to fit the data perfectly. Note that this is a combined assumption about the algorithm (for instance the expressivity of its hypothesis class) and the data distribution. If the algorithm is *fully expressive* (able to express any function), and minimizes the training error (it belongs to the class of *empirical risk minimization* (ERM) algorithms), then this condition doesn't put any constraint on \mathcal{D} beyond being noiseless.

For the realizable case, bounds usually take the form

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\epsilon(\mathcal{A}(S)) \leq \frac{f_{\mathcal{A}}(S)}{m} \right] \geq 1 - \delta \quad (4.4)$$

where we have omitted (without loss of generality), the exponent α , because realizable bounds have $\alpha = 1$ in most cases.

Statistical learning theory has focused on these two classes of assumptions, because they are considered to be minimal, and cover most cases of interest. However, other assumptions on \mathcal{D} are sometimes used in more advanced analyses, typically involving data-dependent and non-uniform bounds (see below). As an unusual example, one can consider the bound in Theorem 2.3 in Shawe-Taylor et al. [1998], which could be combined with an assumption on the training error being smaller than a non-zero value, to obtain a type of generalization to the realizable bounds.

Finally, one can make very explicit assumptions on \mathcal{D} , by assuming a Bayesian prior distribution over data distributions. Although this is currently less common in the deep learning theory literature, some interesting results on these lines have recently been developed (see Section 4.3.5 and Bordelon et al. [2020]).

4.2.3.2 According to assumptions on \mathcal{A}

As for \mathcal{D} , supervised learning theory often makes very minimal assumptions on the learning algorithm \mathcal{A} , though there are also a rich set of algorithm-dependent analyses. The minimal

assumptions made on \mathcal{A} is that it outputs predictors within a set of predictors called the *hypothesis class* \mathcal{H} . As this is the minimal assumption often made, we will refer to it as “algorithm-independent” to distinguish it from the approaches which make stronger assumptions. We thus classify bounds on the following two classes.

Algorithm-independent bounds. These bounds only assume that $\mathcal{A}(S) \in \mathcal{H}$, for all S ². Furthermore, they only include bounds where the capacity $f_{\mathcal{A}}(S)$ can only depend on $h = \mathcal{A}(S)$ and S , and not in a general way on \mathcal{A} ³. This definition means that the bound must bound the generalization gap or the generalization error⁴ for the worst case predictor h from a hypothesis class \mathcal{H} . They have the general form

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\forall h \in \mathcal{H}, D(\epsilon(h), \hat{\epsilon}(h)) \leq \frac{f(S, h)}{m^\alpha} \right] \geq 1 - \delta \quad (4.5)$$

Algorithm-independent bounds are commonly classified in two classes, according to the dependence of the capacity $f(h, S)$ on h :

- **Uniform convergence bounds** (or *uniform bounds* for short) are algorithm-independent bounds where the capacity is independent of h . This includes bounds like VC dimension (Section 4.3.1.1) and Rademacher complexity bounds (Section 4.3.1.2).
- **Non-uniform convergence bounds** (or *non-uniform bounds* for short) are algorithm-independent bounds where the capacity depends on h . Common examples include bounds for structural risk minimization (section 4.3.2.1).

Algorithm-dependent bounds. This type of bounds generalize the above class of bounds by considering stronger, or more general, assumptions on \mathcal{A} . We can express this general class

²or for all S within a set of S which have support $1 - \delta$, as in Nagarajan and Kolter [2019]

³We put this constraint because otherwise there wouldn't be any useful distinction with algorithm-dependent bounds. We could just take a set of algorithm-dependent bounds that cover all algorithms with outputs in \mathcal{H} , and make an “algorithm-independent bound”. We choose the definition to avoid this possibility.

⁴Under these definitions, realizable bounds should technically be algorithm-dependent, because they assume something beyond the hypothesis class of the algorithm – they assume that the algorithm is ERM. However, we will still refer to bounds that only add this extra assumption as algorithm dependent, as the ERM assumption may be considered as being relatively weak. However, this distinction becomes important when discussing some subtleties, like those in Section 4.2.3.4.

of bounds as

$$\forall \mathcal{A} \in \mathfrak{A}, \forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[D(\epsilon(\mathcal{A}(S))), \hat{\epsilon}(\mathcal{A}(S))) \leq \frac{f_{\mathcal{A}}(S)}{m^\alpha} \right] \geq 1 - \delta \quad (4.6)$$

where \mathfrak{A} is a set of algorithms which represents our assumptions on \mathcal{A} . An example of this class of bounds are stability-based bounds (Section 4.3.3.1), which rely on the assumption that the algorithm's output $\mathcal{A}(S)$ doesn't depend strongly on S . The case where $\mathfrak{A} = \{\mathcal{A}\}$ corresponds to the analysis of a particular algorithm. We will refer to this special case as *algorithm-specific bounds*. However, in almost all most cases analyses of particular algorithms really rely only on a subset of the properties of the algorithm, so that the results often apply more generally than to a specific algorithm.

4.2.3.3 According to dependence of capacity on S

One more classification that is commonly done on bounds is according to the dependence of the capacity on the training dataset S . For any of the classes of bounds discussed above, we further distinguish the following two classes of bounds.

Data-independent bounds are bounds in which the capacity does not depend on S . Common examples include VC dimension bounds, and the simple (algorithm-independent) bounds based on SRM (e.g. Equation (4.12), and Equation (4.14)).

Data-dependent bounds are bounds in which the capacity depends on S . Common examples are Rademacher complexity bounds. The PAC-Bayes bounds for Bayesian algorithms (like the one we present on Theorem 4.4.1) is another example.

Note that in this chapter we do not consider explicitly data-distribution-dependent bounds, because we only consider bounds that depend on quantities derivable from S . However, we will consider the behaviour of bounds under different assumptions on the data distribution \mathcal{D} . In particular, we could look at the expected value of the bound, or its whole distribution, for different \mathcal{D} . This is the sense in which we will discuss “data distribution dependence” in this chapter. Note that only bounds that depend on S , called *data-dependent bounds*, can depend on the data distribution in this sense.

4.2.3.4 Non-uniform bounds and algorithm-dependent bounds

A common way to get algorithm-dependent bounds is to start with non-uniform convergence bounds, and then make further assumptions on the algorithm which restrict the set of h that may be output for a given S . We can then use the maximum value of the bound among those h as an algorithm-dependent bound valid for algorithms which satisfy the assumptions. Note that this makes the bound automatically data-dependent. This can be expressed by bounds of the form

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\forall h \in \mathcal{H}(S), D(\epsilon(h), \hat{\epsilon}(h)) \leq \frac{f(S, h)}{m^\alpha} \right] \geq 1 - \delta \quad (4.7)$$

where $\mathcal{H}(S)$ is a set which includes $\mathcal{A}(S)$ for the class of algorithms \mathfrak{A} which we are considering. This is what is done, for example, for margin bounds (section 4.3.2.1) for max-margin classifiers, like SVMs, where we assume the algorithm will output an h which maximizes margin, and then plug that condition into a non-uniform margin bound (by setting $\mathcal{H}(S)$ to be the set of max margin classifiers for S).

Non-uniform bounds are often designed with a particular assumptions on the algorithm and dataset in mind. This is because the value of non-uniform bounds depends on both \mathcal{A} and \mathcal{D} . This in turn means that the notion of optimality for non-uniform (and algorithm and data-dependent bounds in general) should also depend on \mathcal{A} and \mathcal{D} as argued in Section 4.3.6.

In the applications to deep learning theory we study, non-uniform convergence bounds are only used as a way to obtain algorithm-dependent bounds, though we still present the fundamental non-uniform bounds in section 4.3.2.1 as useful background for the algorithm-dependent bounds based on them. We in fact divide the algorithm-dependent bounds into those that are based on non-uniform convergence and those that are not. Apart from shedding light into the origin of the different bounds, this will be a useful distinction on the light of recent results from Nagarajan and Kolter [2019], (see also their Appendix G.4) who showed that, for SGD-trained networks, the tightest⁵ double-sided algorithm-dependent bounds for which the capacity is only allowed to depend on the output of the algorithm $\mathcal{A}(S)$ and has no other data-dependence beyond this, give loose bounds for certain data distributions. The intuition behind this result is that such bounds can encode little information about the algorithm beyond

⁵“Tightest” in their analysis refers to the fact that their bound is specific for the particular algorithm they study (SGD-trained DNNs), and the particular dataset (which was synthetically constructed)

the hypothesis class, as they do can not explicitly capture the dependence of $\mathcal{A}(S)$ on S .

Most algorithm-dependent bounds derived from non-uniform bounds that we will study are based on data-independent non-uniform bounds. This fact automatically makes the algorithm-dependent bounds have a capacity that only depends on $\mathcal{A}(S)$, and so would suffer from the limitations pointed out in Nagarajan and Kolter [2019] also show that standard PAC-Bayes bounds based on the general KL-divergence bound (Equation (4.15)) also suffer from these limitations.

There are several ways in which the limitations in Nagarajan and Kolter [2019] may be avoided. 1) only apply the bound to data distributions where their analysis does not apply 2) allow the bound to depend on the data in other ways than via h (e.g. by having explicit dependence on S as in Shawe-Taylor et al. [1998]) 3) bound the risk difference between $\mathcal{A}(S)$ and an h in a class for which uniform convergence works [Negrea et al., 2019] 4) consider non-double-sided bounds. Bounds derived from an analysis assuming realizability can satisfy 2) and 4). They can satisfy 2) because they only guarantee convergence on a hypothesis class which depends on the data (the set of $h \in \mathcal{H}$ with 0 error). Bounds considered by Nagarajan and Kolter [2019] even if they can depend on h , they are worst-case over training sets. However, analysis which assume realizability can bound the generalization gap only for those training sets S where $\hat{\epsilon}(h) = 0$. Realizable bounds can satisfy 4) because they can use the one-sided version of the Chernoff bound (for 0 mean), as can be seen for example the Corollary 2.3 in Shalev-Shwartz and Ben-David [2014]. However, note that realizable bounds obtained by simply setting $\hat{\epsilon}(h) = 0$ in an agnostic bound will not satisfy these properties, because the agnostic bound does not satisfy them. Therefore, only bounds which take full advantage of the realizable assumption may avoid the limitations in Nagarajan and Kolter [2019].

The PAC-Bayes bound we present in Section 4.4 is based on a realizable analysis using one-sided bounds and thus avoids the limitations of double-sided bounds. In fact, our bound holds with high probability over the Bayesian posterior, rather than universally over a whole family of hypothesis-dependent posteriors, as usually considered by deterministic PAC-Bayes bounds [Nagarajan and Kolter, 2019].

4.2.3.5 Overview of bounds

In the next sections, we will describe the major families of generalization error bounds that have been applied to DNNs. Here we give a high-level overview of where different general classes of bounds found in the literature fit within the classification introduced above. Here we present the classification so that roughly top to bottom (within one level of the hierarchy) goes from weaker to stronger assumptions.

- Algorithm-independent bounds (section 4.3.1)
 - Uniform convergence bounds
 - * Data-independent uniform convergence bounds: VC dimension bounds (section 4.3.1.1)
 - * Data-dependent uniform convergence bounds: Rademacher complexity bounds (section 4.3.1.2)
- Algorithm-dependent bounds (section 4.3.2)
 - Algorithm-dependent bounds based on non-uniform convergence: margin bounds (4.3.2.1), sensitivity-based bounds (section 4.3.2.1), other PAC-Bayes bounds (section 4.3.2.2)
 - Other algorithm-dependent bounds (section 4.3.3)
 - * Data-independent algorithm-dependent bounds: uniform stability bounds and compression bounds (section 4.3.3.1)
 - * Other data-dependent algorithm-dependent bounds: non-uniform stability bounds (section 4.3.3.1)

Given this hierarchy of the main types of bounds, we next turn to a comparison of their performance. As expected, the overall empirical performance of the bounds improves as more assumptions are added.

4.3 Comparing existing bounds against desiderata

In this section we use the framework from Section 4.2 to organise a discussion on how these fare against the desiderata proposed in Section 4.1. We use a X when there is strong evidence that bounds in this family fail to satisfy most important aspects of the desiderata, \checkmark when there is strong evidence that bounds in the family satisfy most important aspects of the desiderata, and \bullet otherwise. We are aware these are not formally defined notions, and the marks should just be taken as an aid for the reader.

4.3.1 Algorithm-independent bounds

4.3.1.1 Data-independent uniform convergence bounds: VC dimension

Vapnik and Chervonenkis considered *data-independent* uniform convergence bounds, where $f_{\mathcal{H}}(S)$ doesn't depend on S . They proved that the optimal bound (up to a multiplicative fixed constant) of this form for the generalization gap, for the case of binary classification is

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\sup_{h \in \mathcal{H}} |\epsilon(h) - \hat{\epsilon}(h)| \leq C \sqrt{\frac{\text{VC}(\mathcal{H}) + \ln \frac{1}{\delta}}{m}} \right] \geq 1 - \delta \quad (4.8)$$

for some constant C , and where $\text{VC}(\mathcal{H})$ is a combinatorial quantity called the Vapnik-Chervonenkis dimension [Shalev-Shwartz and Ben-David, 2014], depending on \mathcal{H} alone. In the realizable case, they also proved that the optimal realizable data-independent uniform bound is

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\sup_{h \in \mathcal{H}_0(S)} \epsilon(h) \leq C \frac{\text{VC}(\mathcal{H}) + \ln \frac{1}{\delta}}{m} \right] \geq 1 - \delta \quad (4.9)$$

for some constant C , and where $\mathcal{H}_0(S)$ is the set of all $h \in \mathcal{H}$ with zero training error on S . The particular realizability assumption here is that \mathcal{D} should be such that for all S , $\mathcal{H}_0(S)$ is non-empty.

How does this bound do at the desiderata?

- **D.1 X** The bound is data-independent by construction, and therefore its value is the same for any data distribution or training set.

- **D.2 ✗** The bound decreases with training set size, but at a rate $O(1/m)$ which is independent of the dataset, unlike what is observed in practice ($O(m^{-\alpha})$ for a range of m , for α often significantly smaller than 1).
- **D.3 ✗** The VC dimension can capture differences in architectures. However, it doesn't appear to capture the correct trends. For example, VC dimension grows with the number of parameters [Harvey et al., 2017], while for neural networks the generalization error tends to decrease (or at least not increase) with increased overparametrization [Neyshabur et al., 2019].
- **D.4 ✗** The bound is only dependent on the algorithm via the hypothesis class. Therefore it won't capture any algorithm-dependent behaviour except for regularization techniques that restrict the hypothesis class.
- **D.5 ✗** The VC dimension of neural networks used in modern deep learning is typically much larger than the number of training examples [Zhang et al., 2017a], thus leading to vacuous VC-dimension bounds.
- **D.6 ✓** Although computing the exact VC dimension of neural networks is intractable, there are good approximations and bounds which are easily computable [Harvey et al., 2017].
- **D.7 ✓** The VC dimension offers a rigorous bound with minimal assumptions. Therefore, its guarantees are rigorously applicable to many cases.

Because the VC dimension bound is optimal among data-independent algorithm-independent uniform bounds, to make progress on the unchecked desiderata, we should consider other types of bounds. Next, we consider data-dependent algorithm-independent uniform convergence bounds. We will also look at data-independent algorithm-dependent bounds in Section 4.3.3.1 when we look at stability bounds.

4.3.1.2 Data-dependent uniform convergence bounds: Rademacher complexity

The theory of empirical processes provides a data-dependent uniform convergence bound (when D is the absolute value function). It is given by

$$\mathbf{P}_{S \sim \mathcal{D}^m} \left[\sup_{h \in \mathcal{H}} |\epsilon(h) - \hat{\epsilon}(h)| \leq 2\mathcal{R}(L \circ \mathcal{H} \circ S) + 4c\sqrt{\frac{2 \ln(4/\delta)}{m}} \right] \geq 1 - \delta \quad (4.10)$$

where c is a constant, and $\mathcal{R}(L \circ \mathcal{H} \circ S)$ is the Rademacher complexity of the set of vectors $L \circ \mathcal{H} \circ S = \{(L(x_i, h(x_i)))_{i=1}^m : h \in \mathcal{H}\} \subseteq \mathbb{R}^m$ where x_i are the input points in S [Bartlett and Mendelson, 2002, Shalev-Shwartz and Ben-David, 2014]. This bound has a lower bound which matches it up to a constant and up to an additive term of $O(1/\sqrt{m})$ [Bartlett and Mendelson, 2002, Koltchinskii, 2011]. Therefore whether this bound is the optimal data-dependent uniform generalization gap bound up to a constant, depends on the rate of the Rademacher complexity with m . Bounds on the Rademacher complexity tend to be of $O(1/\sqrt{m})$ but they often dominate the second term, suggesting the bound in Equation (4.10) may often be close to optimal.

Note that this bound is actually a looser version of a bound which is *data-distribution-dependent*, rather than data-dependent [Shalev-Shwartz and Ben-David, 2014].

$$\mathbf{P}_{S \sim \mathcal{D}^m} \left[\sup_{h \in \mathcal{H}} |\epsilon(h) - \hat{\epsilon}(h)| \leq \mathbf{E}_{S' \sim \mathcal{D}^m} [2\mathcal{R}(L \circ \mathcal{H} \circ S')] + c\sqrt{\frac{2 \ln(4/\delta)}{m}} \right] \geq 1 - \delta \quad (4.11)$$

How do these bounds do at the desiderata? Note that there is a big overlap with the highly related margin-based bounds in Section 4.3.2.1, but we still comment on these for pure Rademacher complexity-based bounds for completeness.

- **D.1 ✗** The bound is data-dependent, and could capture some dependence in the dataset. However, it only depends on the distribution over inputs, and therefore it can't depend on the complexity of the target function, when the input distribution is fixed. This is unlike real neural networks which generalize worse when the labels are corrupted [Zhang et al., 2017a].
- **D.2 ✗** Bounds on the Rademacher complexity are typically $O(1/\sqrt{m})$, thus not capturing

the behaviour of learning curves. Furthermore, margin-based bounds (which are based on norm-based bounds to the Rademacher complexity), often *increase* with m . See Section 4.3.2.1.

- **D.3 ✗** As Rademacher complexity captures a notion of expressivity, similarly to VC dimension, it often grows with overparametrization and number of layers (see for example, the norm-based bounds on Section 4.3.2.1). It seems unlikely it could capture other architectural differences.
- **D.4 ✗** Like the VC dimension bound, it is only dependent on the algorithm via the hypothesis class. The most studied parameter for an algorithm-dependent hypothesis class is the norms of the weights of a parametrized model. As we comment on Section 4.3.2.1, these bounds appear to anti-correlate with the true error when changing several common optimization hyperparameters.
- **D.5 ✗** Like the VC dimension bound, these bounds are typically vacuous in the overparametrized regime. In fact, it has recently been shown that there are data distributions where the *tightest* double-sided distribution-dependent uniform convergence bounds for several SGD-trained models are provably vacuous [Nagarajan and Kolter, 2019], implying that Equation (4.11), and thus Equation (4.10), are vacuous for that data distribution. Although for other data distributions the bounds may not be vacuous, this work suggest that uniform convergence bounds have some fundamental limitations.
- **D.6 ✓** Same as for VC dimension bounds.
- **D.7 ✓** Same as for VC dimension bounds.

Bounds based on Rademacher complexity are agnostic. The authors are not aware of data-dependent algorithm-independent uniform bound under realizability assumptions. Note that agnostic bounds on the generalization gap are double-sided, meaning that they offer both an upper and lower bound on $\epsilon(h) - \hat{\epsilon}(h)$. The limitations of double-sided uniform convergence bounds is analysed in Nagarajan and Kolter [2019]. On the other hand, bounds based on the realizability assumption tend to be one-sided, as they offer just an upper bound on $\epsilon(h)$. These

bounds tend to be significantly tighter than simply applying the double-sided agnostic bound to the realizable case.

Although some fundamental limitations of VC dimension are overcome by Rademacher bounds, the bounds that currently exist for neural networks based on Rademacher complexity have very similar problems to those based on VC dimension. The question of whether some of the desiderata are partially achievable with better analysis is still open, as far as the authors know. However, the fact that the bounds are vacuous suggest they are probably not capturing the origin of generalization, reducing our confidence that it may be able to capture the correct trends.

Regardless of the exact extent of the limitations of Rademacher complexity bounds, the fact that they may be optimal among uniform generalization gap bounds, suggests that to overcome the limitations highlighted here we may need to look at data-dependent non-uniform and algorithmic-dependent bounds. However, as we discuss in Section 4.3.6 for data-dependent bounds, there isn't in general a unique notion of optimality, so that whether one bound is tighter than another depends on what prior assumptions are made on the data distribution.

4.3.2 Algorithm-dependent bounds

Now we will look at the major classes of algorithm-dependent bounds and how they fare at satisfying the desiderata. We will also focus on realizable bounds (which assume zero training error), because modern deep learning often works in this regime [Zhang et al., 2017a]. For many of the recent bounds for deep learning, the lack of experiments means that we can't conclusively answer whether they satisfy several of the desiderata. We hope that future work can fill these gaps.

4.3.2.1 Algorithm-dependent bounds based on non-uniform convergence

We start by looking at algorithm-dependent bounds derived from non-uniform convergence bounds (see also Section 4.2.3.4). We begin by presenting in the next section the basic types of non-uniform bounds, before seeing the two main applications for deep learning in the next two sections: margin bounds, and sensitivity-based bounds. We will also briefly comment on some

other approaches to apply PAC-Bayes to deep learning that have been proposed.

Basic non-uniform convergence bounds and structural risk minimization

The simplest and most fundamental idea to make non-uniform bounds is related to a learning technique called structural risk minimization (SRM) developed by Vapnik and Chevonenkis [Vapnik, 1995] and can be seen as an extension to the original PAC-bound for finite hypothesis classes developed by Leslie Valiant. The idea is that rather than using a union bound in the derivation of the PAC bound (see e.g. corollary 2.3 in Shalev-Shwartz and Ben-David [2014]), we use a weighted union bound over the hypothesis class. Doing this, one can prove that for any countable hypothesis class \mathcal{H} and any distribution P [McAllester, 1998, Shalev-Shwartz and Ben-David, 2014]

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\forall h \in \mathcal{H} \text{ such that } \hat{\epsilon}(h) = 0, \epsilon(h) \leq \frac{\ln \frac{1}{P(h)} + \ln \frac{1}{\delta}}{m} \right] \geq 1 - \delta. \quad (4.12)$$

This bound tells us that if we have a learning algorithm and a problem for which we think some functions h are more likely to be learned than others, we can use this prior knowledge to obtain tighter bounds, by choosing a P that assigns a higher value to the h we think are more likely. Choosing a P amounts to “betting” that some h are more likely to appear. If we are wrong and we get h with low P this bound will be worse than the standard finite-class PAC bound [Valiant, 1984]. One can also intuitively understand this bound as a “soft” version of the finite-class PAC bound, where we can effectively reduce the size of the hypothesis class by placing more probability mass on some hypotheses.

We can study how well this bound works, for a particular choice of P , by assuming a distribution over data distributions \mathcal{D} , which we call the prior Π as in the previous section. For example, this may be fully supported on a single \mathcal{D} if we know the distribution fully (perhaps we are looking only at the images in CIFAR10). In more real-world settings, we will have uncertainty over what the true distribution is, but we may believe that certain distributions (e.g. simpler ones) are more likely. For a stochastic learning algorithm which for training set S outputs predictors with a probability $Q(h|S)$ (called the *posterior*⁶), we can compute the

⁶which need not be the Bayesian posterior

expected value of the capacity in the bound Equation (4.12) for a prior over \mathcal{D} , Π , and a certain P as

$$\mathbf{E}_{\mathcal{D} \sim \Pi, S \sim \mathcal{D}^m, h \sim Q(h|S)} [-\log P(h)] = \mathbf{E}_{h \sim \tilde{Q}} [-\log P(h)] \equiv H(\tilde{Q}, P) \quad (4.13)$$

where $H(\cdot, \cdot)$ is the cross-entropy, and $\tilde{Q}(h) = \mathbf{E}_{\mathcal{D} \sim \Pi, S \sim \mathcal{D}^m} [Q(h|S)]$. The second equality follows from the definition of cross-entropy. We can immediately see that the optimal bound of this form, for a given prior Π and algorithm with posterior $Q(h|S)$ is obtained by the choice $P = \tilde{Q}$, in which case we obtain $H(\tilde{Q})$ where H is the entropy. This calculation formalizes the intuition that we should choose P to be as close as possible to the probabilities of obtaining different h for the algorithm and task at hand.

Furthermore, we can consider the case where $Q(h|S)$ is given by the Bayesian posterior for prior Π . In this case, $\tilde{Q} = \tilde{\Pi}$, where $\tilde{\Pi}$ is the prior distribution over predictors h , obtained by marginalizing over input distributions \mathcal{D}_X ⁷. We see that the average capacity in this case is $H(\tilde{\Pi})$. This is conceptually similar to the No Free Lunch theorem, in that it tells us that for the optimal algorithm, the bound only guarantees good generalization if we make enough assumptions about the data distribution (which corresponds to a low entropy $H(\tilde{\Pi})$). We note, however, that the Bayesian posterior may not give the optimal value of the bound, as can be seen from the fact that $H(\tilde{Q}, P)$ is always lowered by making $Q(h|S)$ deterministic (as it would lower $H(\tilde{Q})$), while the Bayesian posterior is not deterministic in general.

The problem with the basic SRM bound (Equation (4.12)) is that it does not capture the idea that some functions may be more similar to others, which quantities like VC dimension and Rademacher complexity do capture⁸. The generalized version of SRM bound which we present below has the advantage of being non-uniform but being able to capture some notion of similarity among function within the subclasses \mathcal{H}_i .

A more commonly-used extension of the basic SRM bound, considers dividing the (now potentially uncountable) hypothesis class \mathcal{H} into a countable set of (usually nested) subclasses \mathcal{H}_i , $i \in \mathbb{N}$, such that $\bigcup_i \mathcal{H}_i = \mathcal{H}$. The result is that for any distribution P over \mathbb{N} [Shalev-Shwartz

⁷Note that, unless we restricted to the noiseless case, these would be stochastic predictors corresponding to $\mathcal{D}_{Y|X}$ as defined in Section 4.2.1

⁸For example the VC dimension of a set of functions which are very similar to each other will typically be lower than a set of very dissimilar functions

and Ben-David, 2014],

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\forall i \in \mathbb{N} \ \forall h \in \mathcal{H}_i \text{ such that } \hat{\epsilon}(h) = 0, \ \epsilon(h) \leq \frac{\ln \frac{1}{P(i)} + f_i(S) + \ln \frac{1}{\delta}}{m} \right] \geq 1 - \delta \quad (4.14)$$

where $f_i(S)$ is any (potentially data-dependent) capacity for class \mathcal{H}_i which guarantees uniform convergence within \mathcal{H}_i (for example, a bound on its VC dimension or Rademacher complexity). Results of this form are proven in Shawe-Taylor et al. [1998] and Shalev-Shwartz and Ben-David [2014].

We can also compute the expected value of the bound Equation (4.14), analogously to Equation (4.12). For the numerator of the bound (ignoring the confidence term), we obtain $H(\tilde{Q}', P) + \mathbf{E}_{i \sim \tilde{Q}'} [f_i(S)]$, where $\tilde{Q}'(i) \equiv \mathbf{P}_{h \sim \tilde{Q}} [\text{class}(h) = i]$ and $\text{class}(h)$ represents the index of the subclass $\mathcal{H}_{\text{class}(h)}$ to which h belongs. Analogously to before, the optimal value of P is given by \tilde{Q}' , and the Bayesian posterior will in general not result in the optimal average value of the bound.

One shortcoming of Equation (4.14) is that the decomposition of \mathcal{H} into \mathcal{H}_i has to be defined a priori, that is, it cannot depend on S . Shawe-Taylor et al. [1998] proposed an extension to the SRM framework which addressed this shortcoming, and defined a potentially infinite hierarchy of subclasses $\mathcal{H}_1(S) \subseteq \mathcal{H}_2(S) \subseteq \dots$ which could depend on the data S . This framework includes as a special case the margin bounds we will see in Section 4.3.2.1.

Shawe-Taylor and Williamson [1997] applied the data-dependent SRM framework to obtain bounds where the capacity was related to the volume in parameter space of a parametrized model of a sphere contained within the set of parameters producing zero training error. This work inspired the development of the first PAC-Bayes bounds in McAllester [1998]⁹. These bounds apply for stochastic learning algorithms, and bound the expected value of the generalization error under the posterior $Q(h|S)$, uniformly over posteriors. The standard form of the general

⁹Although Shawe-Taylor and Williamson [1997] is often cited as a precursor to PAC-Bayes Shawe-Taylor [2019], it offers a distinct analysis (for example, it gives deterministic bounds rather than bounds on expected error), which as far as the authors know hasn't been shown to necessarily give stronger or weaker bounds than PAC-Bayes, and it hasn't been applied to neural networks.

PAC Bayes bound was proven by Maurer [2004] and states, for any distribution P over \mathcal{H} ,

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\forall Q \ KL(\mathbf{E}_{h \sim Q}[\epsilon(h)], \mathbf{E}_{h \sim Q}[\hat{\epsilon}(h)]) \leq \frac{KL(Q||P) + \ln \frac{1}{\delta} + \ln(2m)}{m-1} \right] \geq 1 - \delta \quad (4.15)$$

where $KL(Q||P)$ is the KL-divergence between Q and P , and for $a, b \in [0, 1]$ $KL(a, b) \equiv -a \ln b - (1-a) \ln(1-b)$.

This bound can be seen to generalize the SRM with data-dependent hierarchies of Shawe-Taylor et al. [1998], where instead of ‘‘hard’’ subdivisions of \mathcal{H} into \mathcal{H}_i , we consider all possible distributions Q on \mathcal{H} . $KL(Q, P)$ is analogous to $\ln \frac{1}{P(i)}$ in Equation (4.14) in that it very roughly measures how much of the total probability mass of P is in the high probability region of Q . The term which penalizes ‘‘classes’’ of h which are too diverse, analogously to $f_i(S)$, is $\mathbf{E}_{h \sim Q}[\hat{\epsilon}(h)]$, the average training error, because this is only small if the functions agree on S , which will only happen with high probability if they are sufficiently similar. These analogies between the data-dependent SRM bounds and PAC-Bayes are only intuitive, but the authors conjecture a more formal connection might be possible.

The PAC-Bayes bound in Equation (4.15) is one of the most general non-uniform data-dependent bounds, and its different applications give rise to sensitivity-based bounds (Section 4.3.2.1) among others (Section 4.3.2.2). In fact, margin bounds can also be derived from PAC-Bayes [Langford and Shawe-Taylor, 2003].

We don’t comment on the desiderata for SRM and the general PAC-Bayes bound, because they provide a general framework for the non-uniform data-dependent bounds we explore next. We will comment on the desiderata for these bounds individually.

Norm-based and margin bounds

A popular method to obtain data-dependent non-uniform bounds is the method of margin-based bounds. These bounds usually start with *norm-based bounds* which bound the Rademacher complexity of sub-classes of a hypothesis class parametrized by a parameter vector $w \in \mathcal{W}$ (*weights*) corresponding to balls where w has a bounded norm. One can then either express the bound under an assumption on the weight norms, or have the bound depend on the weight norms. The later case is done by applying an SRM-like bound to the family of hypothesis

sub-classes corresponding to different weight norms. For *margin bounds*, the bound is applied to a *margin loss*, which upper bounds the 0-1 loss. The result is a bound on the generalization error which depends on a new data-dependent property called *margin*, which measures how confidently the classifier is classifying examples. The bound, in the agnostic case, has the general form, for any margin $\gamma > 0$ [Shalev-Shwartz and Ben-David, 2014],

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[\forall w \in \mathcal{W}, \epsilon(w) \leq \hat{\epsilon}_\gamma(w) + c_1 \frac{\lambda(w)}{\gamma \sqrt{n}} + c_2 \sqrt{\frac{2 \ln(4/\delta)}{m}} \right] \geq 1 - \delta \quad (4.16)$$

where c_1 and c_2 are constants (that may depend on the algorithm but not on h or S), $\lambda(w)$ is a *capacity measure* which usually measures the norm of the parameters w , and the margin error is defined as $\hat{\epsilon}_\gamma(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}[h(x_i)y_i < \gamma]$, for a hypothesis h . Note that we usually apply this to real-valued hypotheses where $\mathcal{Y} = \mathbb{R}$, and for which the classification error is defined as $\epsilon(h) = \mathbf{P}_{(x,y) \sim \mathcal{D}}[h(x)y < 0]$, where $y \in \{-1, 1\}$ for binary classification. We abuse notation by writing $\epsilon(w)$ and $\hat{\epsilon}_\gamma(w)$ for the quantities evaluated at the hypothesis corresponding to parameter w . One can also make a bound that holds (non-uniformly) for all values of γ by applying a weighted union bounds over discretized values of γ [Shalev-Shwartz and Ben-David, 2014].

The margin loss $\hat{\epsilon}_\gamma(w)$ measures the amount of miss-classification errors plus some examples which were classified correctly but with low confidence (measured by $h(x)$ being smaller than γ). In the case of linear models, where $h(x) = w \cdot x$ and $\lambda(w) = |w|$, the ratio $\gamma' = \lambda(w)/\gamma$ measures a *geometric margin*, and the margin loss measures the number of examples that are not on the right side of the classification boundary by a distance greater than γ' . Support vector machines are a famous example of an algorithm trying to maximize this geometric margin [Cortes and Vapnik, 1995].

For neural networks, margin bounds were originally developed based on the analysis of their fat-shattering dimension Bartlett [1997, 1998]. These bounds depend on the l_1 norm of the weights, and thus typically grows with overparametrization. The bounds also grow exponentially with depth. More recently, more complex norms of the weights were studied, as well as using the Lipschitz constant of the network as the capacity measure for margin-bounds Bartlett et al. [2017], Neyshabur et al. [2018a], Golowich et al. [2018], Neyshabur et al. [2018b], Barron and

Klusowski [2019]. These bounds have shown some correlation with the complexity of the data, but suffer from their (implicit or explicit) dependence on width and depth, and are vacuous Neyshabur et al. [2017, 2015a], Arora et al. [2018]. They also show negative correlation with the generalization error when changing certain training hyperparameters Jiang et al. [2019].

We now summarize the strengths and weaknesses of margin-based bounds on our desiderata. Note that most of our conclusions are based on empirical results on existing margin-based bounds, and may not be fundamental to the margin-based approach itself.

- **D.1 ✓** Margin-based bounds have shown to correlate with the true error when comparing CIFAR versus MNIST, and when comparing uncorrupted versus corrupted data (Bartlett et al. [2017], Neyshabur et al. [2017]). The correlation should be explored over a wider range of datasets and quantified more precisely.
- **D.2 ●** The dependence on the training set size m of margin-based bounds depend on how the capacity measure changes with m . In Nagarajan and Kolter [2019], it is shown that several types of the margin-based bounds proposed for deep neural networks, actually increase with training set size! However, in Dziugaite et al. [2020], other norm and margin-based generalization measures are found to correlate well with the error when changing training set size, suggesting that norm and margin-based bounds with this property may be possible.
- **D.3 ✗** While most margin-based bounds increase with layer width, some (l_2 -path norm bound in Neyshabur et al. [2017] and the bound in Neyshabur et al. [2018b]) actually decrease with layer width. Regarding variations in depth, all the proposed bounds increase with number of layers, and most of them do so exponentially Neyshabur et al. [2015a], Arora et al. [2018], Barron and Klusowski [2019]. However, the empirical results in Jiang et al. [2019], Maddox et al. [2020], Dziugaite et al. [2020] show that certain measures (e.g. path-norm) positively correlate with error when varying depth, suggesting that bounds that capture the correct depth dependence may be possible based on these measures. In particular, Maddox et al. [2020] show that the log path-norm correlates with both width and depth significantly better than path-norm.

- **D.4** ● Jiang et al. [2019] show that margin-based bounds that have been proposed to date appear to often *anti*-correlate with the generalization error when changing common optimization hyperparameters (like dropout, learning rate, type of optimizer, etc.). On the other hand, in Dziugaite et al. [2020], we see that some measures like path norm, often predicts well when changing learning rate (but they didn't vary the other hyperparameters studied in Jiang et al. [2019]).
- **D.5** ✗ As far as the authors are aware, all of the margin-based bounds published to date are vacuous Neyshabur et al. [2017, 2015a], Arora et al. [2018].
- **D.6** ✓ Margin-based bounds are often based on a notion of the norm of the weights that is relatively efficiently computable.
- **D.7** ✓ Proposed bounds are based on rigorous theorems with typically weak assumptions.

Bounds based on the neural tangent kernel

It was recently demonstrated that infinite-width DNNs when trained by SGD with infinitesimal learning rate evolve in function space as linear models with a kernel known as the neural tangent kernel (NTK) [Jacot et al., 2018, Lee et al., 2019]. Several works, either inspired by NTK or not, have relied on the linearization of the dynamics for wide neural networks. The bounds are similar to the norm-based bounds in the previous section 4.3.2.1 in that they tend to bound the Rademacher complexity of hypothesis subclasses characterized by a norm, usually the norm of the deviation of the weights from initialiation. The analyses based on NTK are often also able to guarantee *convergence* of the optimizer, so that we can also bound the empirical risk for a sufficiently large number of optimizer iterations. For instance, Arora et al. [2019a] prove that for a two-layer fully connected DNN, for a sufficiently wide neural network, and sufficiently many (full batch) gradient descent steps t , we have

$$\forall \mathcal{D}, \mathbf{P}_{S \sim \mathcal{D}^m} \left[R(h_t) \leq \sqrt{\frac{2\mathbf{y}^T(\mathbf{H}^\infty)^{-1}\mathbf{y}}{m}} + O\left(\sqrt{\frac{\log \frac{m}{\lambda_0 \delta}}{m}}\right) \right] \geq 1 - \delta \quad (4.17)$$

where h_t is the hypothesis learnt by the DNN, \mathbf{y} is the vector of training outputs y_i , \mathbf{H}^∞ is the Gram matrix for the inputs in the training set S , and λ_0 is a lower bound on the eigenvalues of

\mathbf{H}^∞ . See Arora et al. [2019a] (Theorem 5.1) for the full statement of the result. The connection to the norm-based bounds comes from the NTK analysis they carry out, which shows that the Frobenius norm of the deviation of the weights from initializatoin is bounded by $\sqrt{\mathbf{y}^T(\mathbf{H}^\infty)^{-1}\mathbf{y}}$ plus higher order terms. They then use this bound to obtain a data-dependent bound on the Rademacher complexity. NTK analyses could provide tighter bounds on Rademacher complexity than those we saw in section 4.3.2.1, because the NTK likely captures the relation between the parameters and the function the network implements more precisely than the analyses based on bounding the Lipschitz constant or similar quantities.

More recently, Cao and Gu [2019] sharpened Arora et al.’s result with a similar bound, but which applies to networks with any depth trained with SGD. In their bound, \mathbf{H}^∞ is replaced by the NTK matrix, which they show gives a tighter bound.

- **D.1 ✓** The NTK-based capacity in Arora et al. [2019a] was shown to increase with amount of label corruption on MNIST and CIFAR, and on MNIST for the capacity measure in Cao and Gu [2019].
- **D.2 ●** The authors are not aware of any work studying the dependence of NTK-based bounds on m . The $O(m^{-1/2})$ is not necessarily indicative because the numerator in the bound likely has non-trivial dependence on m .
- **D.3 ✗** Like norm-based margin bounds, current NTK-bounds grow with depth [Cao and Gu, 2019]. On the other hand, they show very little dependence on network width, for large enough width. This is a property of NTK analyses which matches empirical observations well.
- **D.4 ●** The authors are not aware of any work studying the dependence of NTK-bounds on the optimization algorithm. Most analyses focus on vanilla versions of SGD with specific hyperparameter choices which help the theoretical analysis. Other optimization algorithms have been shown to have NTK limits (e.g. momentum in Lee et al. [2019]), but we are not aware of generalization bounds for these.
- **D.5 ●** The bounds in Cao and Gu [2019] are non-vacuous (at least the dominant term), but they are not very tight (with values close to 1 above 20% label corruption).

- **D.6 ✓** The NTK of fully connected networks has an analytical form which is efficiently computable [Lee et al., 2019]. However, for more complex architectures, it may be necessary to estimate the NTK limit (when it exists – see Yang [2019a]) via Monte Carlo [Novak et al., 2019b].
- **D.7 ✓** Proposed bounds are based on rigorous theorems, though the assumptions on the algorithms and the width are sometimes hard to match in practice (too large width, or too small learning rate).

Sensitivity-based bounds

Many generalization error bounds recently developed and applied to deep learning are based on the idea that neural networks whose outputs (or loss function values) are robust to perturbations in the weights may generalize better. This is linked to the observed phenomenon that flatter minima empirically generalize better than sharper minima [Hochreiter and Schmidhuber, 1997a, Hinton and van Camp, 1993, Zhang et al., 2018, Keskar et al., 2016]. At an informal level, it has been argued that the reason for this correlation is that flatter minima may correspond to simpler functions [Hochreiter and Schmidhuber, 1997a, Wu et al., 2017]. In particular, Hochreiter and Schmidhuber [1997a] link flatness to generalization via the idea of *minimum description length* (MDL) [Rissanen, 1978]. MDL generalization bounds are formally equivalent to the simple SRM bound in Equation (4.12), where $-\log P(h)$ is often interpreted as the length of the string representing h under some prefix-free code [Shalev-Shwartz and Ben-David, 2014].

A more sophisticated argument linking flatness to generalization is found in the data-dependent SRM analysis of Shawe-Taylor and Williamson [1997]. As we mentioned in Section 4.3.2.1, they proved generalization bounds where the capacity was mainly controlled by the volume of a region (which they took to be a ball) in weight space in which the training error was zero. A larger volume corresponds to a flatter minimum. Note that one difference with previous work is that Shawe-Taylor and Williamson [1997] define flatness in terms of the classification error, rather than the loss function (for which one typically uses the Hessian as a measure of flatness).

Recent theoretical works studying the link between sensitivity, flatness, and generalization focus on PAC-Bayes analyses Neyshabur et al. [2017], Jiang et al. [2019]. In the PAC-Bayes bound (Equation (4.15)), the $KL(Q, P)$ term is typically larger when the posterior Q has a larger variance, so that it can ‘‘overlap’’ with the prior P more. On the other hand to control the average training error $\mathbf{E}_{h \sim Q}[\hat{\epsilon}(h)]$, we need Q to put most of its weight on regions of low error. If we combine these two considerations, the bound typically predicts best generalisation for large regions of weight space with low error (flat minima). However, as shown in Neyshabur et al. [2017], a more careful look at the PAC-Bayesian analysis suggests that flatness alone is not sufficient to control capacity and should be complemented with some other measure such as norm of the weights. In particular, if Q is taken to be a Gaussian around the weights found after training, and P is taken to be a Gaussian around the origin, then the $KL(Q, P)$ term also grows with the norm of the weights. We can see this in the bound proposed in Neyshabur et al. [2017] which states that, for all \mathcal{D} , with probability at least $1 - \delta$, over $S \sim \mathcal{D}^m$, we have

$$\mathbb{E}_{\nu \sim N(0, \sigma)^n} [R(f_{w+\nu})] - \widehat{R}(f_w) \leq \underbrace{\mathbb{E}_{\nu \sim N(0, \sigma)^n} [\widehat{R}(f_{w+\nu})] - \widehat{R}(f_w)}_{\text{expected sharpness}} + 4 \sqrt{\frac{1}{m} \underbrace{\left(\frac{\|w\|_2^2}{2\sigma^2} + \ln \frac{2m}{\delta} \right)}_{\text{KL}}} \quad (4.18)$$

where $f_w = A(S)$ and w are the function and weights, respectively, produced by the network after training, and f_ν is the function obtained by perturbing the weights of the network, w , by the noise ν . σ is a hyperparameter that can be chosen to take any value greater than 0. The first two terms after the inequality are called *expected sharpness* by Neyshabur et al. [2017], and measures how much the loss increases in average by a perturbation of order σ to the weights. Neyshabur et al. [2017] perform experiments that show that this bound correlates well with the true error when varying data complexity, and number of training examples, but not when varying the amount of overparametrization. By optimizing the posterior of a PAC-Bayes bound conceptually similar to Equation (4.18), Dziugaite and Roy [2017] also obtained bounds on the expected error under weight perturbations. These were noteworthy because they were non-vacuous.

The bounds in Neyshabur et al. [2017], Dziugaite and Roy [2017] are bounding the expected

value of the generalization error under perturbation of the weights, rather than the generalization error of the original network. Obtaining bounds on the later is addressed by recent work on *determinist* PAC-Bayes bounds [Nagarajan and Kolter, 2018]. However their bounds are vacuous, and follow the wrong trends when varying depth and width.

In Jiang et al. [2019], it was found empirically that using a worst-case measure of sharpness, that measures the loss change along the worst weight perturbation of a certain magnitude, very similar to the one proposed in Keskar et al. [2016], gives better correlation with the true error when varying several quantities. In fact it gives the best correlation (and best results in their causal analysis) among the many measures they tested.

Arora et al. [2018] developed another approach to prove generalization bounds based on the robustness of neural networks to perturbations of the weights. They showed that if the effects of perturbing the weights does not grow too much as it propagates through every layer¹⁰, the network could be compressed to a network with fewer parameters, for which a generalization error bound could be given that was tighter than other proposed bounds, although still vacuous. They found that their bound correlates with the true error as it decreased during training. However, this bound has the disadvantage that it applies to the compressed network only, and not to the original network.

Recently, Banerjee et al. [2020] have developed novel deterministic bounds based on a de-randomization of a PAC-Bayes bound. Their bound is also based on the flatness of the minimum found after training (measured by Hessian eigenvalues), and also takes into account the distance moved in parameter space. They show evidence that their bound correlates well with the test error when varying the training set size and label corruption. However, they don't study the tightness of their bound (which depends on certain smoothness constants of a linearized version of the non-linear DNN).

- **D.1 ✓** Neyshabur et al. [2017], Banerjee et al. [2020] show some evidence that their PAC-Bayesian bounds correlates with true error when varying the data complexity. The authors are not aware of similar results for the other measures.
- **D.2 ✓** Neyshabur et al. [2017], Banerjee et al. [2020], Dziugaite et al. [2020] show evidence

¹⁰a condition which they formalized through a series of measurable quantities

that different sensitivity-based PAC-Bayesian bounds correlate with true error when varying the training set size. However, a more quantitative comparison could be done, over more datasets and different architectures.

- **D.3** ● Both the worst-case and expected sharpness measures appear to correlate well with the true error when varying depth. However, only the worst-case sharpness appears to correlate with the error when varying width Neyshabur et al. [2017], Jiang et al. [2019]. Furthermore, Dziugaite et al. [2020] show that although the average correlation with depth and width is good for some PAC-Bayes and sharpness measures, they are not robust, and all of them fail for a significant number of experiments. The bound in Arora et al. [2018] depends on quantities whose dependence on the architecture are hard to predict; however, the bound’s explicit dependence on depth suggests that it may grow linearly with depth, unlike what is observed empirically. Recently, Maddox et al. [2020] show that a measure of flatness known as effective dimensionality correlates better with the error, than PAC-Bayes measures, when varying width and depth, suggesting that it may be a better measure than PAC-Bayes-based flatness measures to understand generalization.
- **D.4** ✓ The worst-case sharpness appears to correlate well with the true error when varying several algorithm hyperparameters, while other sharpness measures correlate a bit worse Jiang et al. [2019]. In Dziugaite et al. [2020], we see some flatness measures indeed correlate well with the error over most experiments.
- **D.5** ● Although the sharpness bounds in Neyshabur et al. [2017], Jiang et al. [2019] are likely vacuous, Dziugaite and Roy [2017] showed that by optimizing the PAC-Bayesian prior over a large family of Gaussians, non-vacuous bounds could be obtained.
- **D.6** ● Some sharpness bounds studied in Jiang et al. [2019] are efficiently computable. However, the more advanced ones like the ones in Dziugaite and Roy [2017] require significant computational expense.
- **D.7** ● The bounds in Neyshabur et al. [2017], Dziugaite and Roy [2017], Arora et al. [2018] are based on rigorous theorems. However, they only apply to either the expected

error under random perturbations of the weights, or to the compressed network. The bounds in Nagarajan and Kolter [2018], Banerjee et al. [2020] applies to the deterministic error of the original network, but may be non-vacuous or not very tight. The worst-case sharpness measure which appears to correlate best with the generalization error Jiang et al. [2019] lacks a rigorous theorem that explains this correlation.

4.3.2.2 Other PAC-Bayes bounds

There are many recent works applying PAC-Bayesian ideas to obtain generalization error bounds in novel ways. Zhou et al. [2018] prove non-vacuous generalization error bounds on compressed networks trained on large datasets. However, their bounds are still very loose, and their correlation with the true error hasn't been studied yet. Dziugaite and Roy [2018] extend the PAC-Bayesian analysis to include data-dependent priors under the assumption that they are close to differentially-private priors. Their bounds are non-vacuous and apply to the expected value of the generalization error after training with SGLD [Welling and Teh, 2011], but they are very computationally expensive, and have only been tested on a small synthetic dataset.

4.3.3 Other algorithm dependent bounds

We now look at other types of algorithm-dependent bounds, not based on non-uniform convergence. The main class of such bounds are stability-based bounds, under which we include compression and algorithmic stability bounds. These bounds include both data-independent and data-dependent bounds. The data-independent bounds will suffer from many of the pitfalls as VC dimension bounds, as DNNs shows generalization for some datasets but not others, while the data-dependent bounds are more promising.

4.3.3.1 Stability-based bounds

Stability-based bounds offer an alternative way to obtain algorithm-dependent bounds, different from the non-uniform convergence SRM-like bounds. In fact, they even allow to obtain data-independent algorithm-dependent bounds. Stability analyses show that if the output of a

learning algorithm depends weakly on the training set, then it can be shown to generalize. One approach to showing this was developed by Littlestone and Warmuth [1986], who developed *compression bounds*. They obtain data-independent bounds on the generalization error for learning algorithms whose output can be computed via a fixed function of only $k < m$ out of the m training examples (which k examples they are can depend on the training sample). For the realizable case, the bound is

$$\mathbf{P}_{S \sim \mathcal{D}^m}[\epsilon(\mathcal{A}(S)) \leq \frac{8k \log(m/\delta)}{m}] \geq 1 - \delta. \quad (4.19)$$

See Shalev-Shwartz and Ben-David [2014] for the formal statement and proof. These bounds are based on the general concept of ‘stability’ described above because if the output of the algorithm only depends on a small subset of the training examples, it means the output is insensitive to changes in most of the outputs. A compression bound has been recently developed for two-layer neural networks trained with SGD on linearly-separable data, based on a proof that in this case SGD converges in a bounded number of non-zero weight updates, which therefore gives a bound on k Brutzkus et al. [2018].

The most common notion of stability, called *algorithmic stability*, was related to generalization by Bousquet and Elisseeff [2002], and considers how sensitive the output of the learning algorithm is to removing a single example from the training sample. Most work on algorithmic stability has focused on the data-independent notion of *uniform stability*, which has been used to obtain data-independent bounds for SGD Hardt et al. [2016], Mou et al. [2018].

Because compression bounds and uniform stability are both data-independent, they can’t capture the crucial data-dependence of generalization in deep learning pointed out by Zhang et al. [2017a]. To solve this, some recent extensions have looked at data-dependent notions of stability. Kuzborskij and Lampert [2017] applied this idea to obtain generalization error bounds for SGD-trained models. They obtain bounds of the form

$$\forall \mathcal{D}, \mathbf{E}_{(S \sim \mathcal{D}^m, A)}[R(\mathcal{A}(S)) - \hat{R}(\mathcal{A}(S))] \leq \mathcal{O}\left(\sqrt{c(R(w_1) - R^*)} \cdot \frac{\sqrt[4]{T}}{m} + c\sigma \frac{\sqrt{T}}{m}\right), \quad (4.20)$$

where the expectation is also taken over the randomness in the algorithm \mathcal{A} (as SGD is a

stochastic algorithm), c is a constant, which is related to the step size of SGD, w_1 is the initial weights of the neural network, R^* is the minimum risk achievable by the hypothesis class of the algorithm, T is the training time, and σ is a bound on the variance of the SGD gradients.

The bound in Equation (4.20) has several limitations which we discuss now. First of all, it is a bound on the expected value of the generalization error, which does not immediately imply a bound that holds with high probability and logarithmic dependence¹¹ on the confidence parameter δ , as the other bounds studied here Shalev-Shwartz et al. [2010], Feldman and Vondrák [2019]. The bound applies for smooth convex losses, but the authors also provide a more complex bound for smooth non-convex losses. The smoothness is an important limitation, as most neural networks in practice use ReLU activations, making the loss surface non-smooth. However, in their experiment they use a CNN with max pooling, which gives a non-smooth loss surface, but their bounds still work well empirically. The bound also requires an estimate of $R(w_1)$. They estimate this with a validation set, which makes the bound not dependent on S alone. However, in practice the initialization is usually random and independent of S , which means $R(w_1)$ could be estimated from the empirical loss on S . The bound can not be directly applied to classification error, which is not Lipschitz, but it can be applied to cross-entropy loss which in turn implies an upper bound on classification error. Perhaps the most serious limitation of this stability bound is that it assumes a single pass over the data $T \leq m$, which is not the usual case in practice as training and generalization error often decrease by training on several passes of the data.

Some recent works extend the data-dependent stability analysis SGD-trained neural networks in different directions. London [2017], Li et al. [2019] combine stability bounds with the PAC-Bayesian approach we will discuss later. Zhou et al. [2019a] prove data-dependent stability bounds that apply to SGD with multiple passes over the data. However, their bound increases with training time (although logarithmically rather than polynomially as in Hardt et al. [2016], Mou et al. [2018]), contradicting the empirical result that generalization error appears to plateau with training time Hoffer et al. [2017]. However, their results have not yet been empirically tested, so that it is hard to evaluate these bounds on the desiderata.

¹¹note that the Markov inequality implies a high probability bound, but it has a polynomial $1/\delta$ dependence on the confidence parameter, which is expected to be far from optimal

- **D.1 ✓** Zhou et al. [2019a] show that their data-dependent stability bound correlates with the amount of label corruption on three different datasets. Li et al. [2019] also show that their *Bayes stability* analysis gives bound that are larger for randomly labelled CIFAR10 than for uncorrupted CIFAR10.
- **D.2 ●** The explicit dependence of many stability bounds on m is given by the classical $1/m$ or $\sqrt{1/m}$. However, data dependent bounds have quantities which may change with m in complicated ways. Furthermore, the bounds for non-convex losses in Kuzborskij and Lampert [2017] have a more complicated data-dependence power law explicit dependence on m . Kuzborskij and Lampert [2017] show a good correlation between their bound and the empirical generalization gap. However, this is only done for one-pass (online) SGD. As far as the authors know, no study has compared the m -dependence of the other bounds with the true error, for the usual case where SGD is trained over multiple passes to reach low training error.
- **D.3 ●** The data-dependent stability bounds depend on empirical quantities of the loss surface and the behavior of SGD, both of which can in principle be affected by the choice of architecture. However, as far as the authors know this dependence hasn't been explored.
- **D.4 ✗** Most stability bounds grow with training time. However, empirically the opposite correlation is found, with longer training time leading to better generalization Hoffer et al. [2017], Jiang et al. [2019]. Charles and Papailiopoulos [2018] show situations in which gradient descent (GD) is not uniformly stable but SGD is. However, whether SGD really generalizes better than GD is still a controversial topic Hoffer et al. [2017].
- **D.5 ●** Many stability bounds Hardt et al. [2016], Mou et al. [2018], Zhou et al. [2019a] grow with training time, and thus will become vacuous for sufficiently large training times. Kuzborskij and Lampert [2017] has shown remarkably tight expected generalization gap bounds, which however only apply to one-pass SGD. These results are very promising, but further empirical analysis, and work on tight bounds for multi-pass SGD is still needed.
- **D.6 ✓** Data-independent stability bounds are typically easy to compute. Data-dependent

ones like the one in Kuzborskij and Lampert [2017] are harder (depending on empirical quantities like the Hessian and gradient sizes), but still applicable to reasonably sized problems

- **D.7** Proposed bounds are based on rigorous theorems. However, these often have assumptions which are not held in common practice (e.g. one-pass over the data, smoothness of loss function, linear separability).

4.3.4 Other bounds and generalization measures

Jiang et al. [2018] have recently empirically studied a measure of generalization based on features of the distribution of margins at different hidden layers. Their measure shows significantly better correlation between the bound and the error as data complexity and architecture is varied than the margin-based measures in Bartlett et al. [2017]. However, they provide limited results looking at the predictive power when changing individual features of the data or architecture, and instead provide an aggregate correlation score when they are all changed simultaneously. The success of the measure explored in Jiang et al. [2018] may be related to the correlation observed in Valle-Pérez et al. [2018], Mingard et al. [2020] between the prior probability of functions for Bayesian DNNs and the critical sample ratio (CSR), which was proposed as a complexity measure in Krueger et al. [2017a]. The critical sample ratio is an aggregate measure of the distances between input points and the decision boundary, like the distribution of input margins in Jiang et al. [2018]. Furthermore the correlation between the prior probability and generalization is established in Valle-Pérez et al. [2018], Mingard et al. [2020], as well as our PAC-Bayes bound in Section 4.4, which helps understand why CSR may correlate with generalization.

Wei and Ma [2019a,b] offer theoretical bounds based on a similar idea to Jiang et al. [2018]. They consider extending the notion of margins to all the layers of the DNN. However, no empirical evaluation is presented.

Recently, and in response to the work of Nagarajan and Kolter [2019], Negrea et al. [2019] propose a method to apply uniform convergence to cases where it previously produced vacuous generalization gap bounds. The idea is to show that an algorithm produces a hypothesis

with generalizatoin and empirical errors close to some hypothesis in a class with a uniform convergence property. This work has yet to be applied to deep learning, but could offer an interesting direction.

Finally, several measures investigated by Jiang et al. [2019], Maddox et al. [2020], Dziugaite et al. [2020], some of which we have discussed in the sections on margin and sensitivity-based bounds, don't yet correspond to rigorous analyses of generalization, but some predict generalization better than existing bounds, and so are promising directions for future more rigorous analyses of generalization.

4.3.5 Generalization error predictions for specific data distributions

There is recent work on predicting generalization error of DNNs by making assumptions on \mathcal{D} , rather than relying on frequentist PAC bounds which typically don't make any assumption on \mathcal{D} (or just assume realizability). In [Bordelon et al., 2020], Bordelon et al. study kernel regression with miss-specified priors, based on the work by Sollich [Sollich, 2002]¹². They then apply this idea to the neural tangent kernel (NTK) of a fully connected network, which approximates the behaviour of SGD-trained DNNs in the infinite width and infinitesimal learning rate limit [Jacot et al., 2018], which seems to work well for finite-width but wide DNNs [Lee et al., 2019]. Bordelon et al. [Bordelon et al., 2020] apply this to MNIST by estimating the NTK eigenspectrum on a sample of MNIST, and then training the DNN on smaller samples, and their predicted generalization error closely follows the observed error of the SGD-trained DNN. As far as the authors are aware this is the first work to accurately predict the generalization error of DNNs based on well-established theory.

One of the limitations of the analysis in Bordelon et al. [2020] is that it relies on knowing what the data distribution is, and in particular what are. This can be estimated by using a sufficiently large sample of the data, but it is not discussed in Bordelon et al. [2020] how big the sample needs to be for the estimate to be accurate. They use a sample larger than the training set, which therefore makes this predictor fall outside the requirements of the kinds of predictos we have been considering (which only depend on S). However, the approach offers an analytical theory of generalization which can help with interpretability and gaining understanding of

¹²A similar analysis from a physics-inspired perspective has been presented in Cohen et al. [2019a]

which properties of a DNN architecture lead to generalization for a particular dataset. The other limitation of the work in Bordelon et al. [2020] is that the analysis only applies for MSE loss, which is not commonly used for classification (though the training with the two losses often result in DNNs with similar learned functions Mingard et al. [2020]).

4.3.6 Optimality of data-dependent bounds

We saw that VC dimension bounds are provably optimal (up to a constant) among the set of data-independent algorithm-independent uniform bounds. The question of optimality is more difficult for the other types of bounds. As we will see, for data-dependent bounds the optimal bound may depend on a chosen prior over data distributions, as well as the algorithm. We don't consider here notions of optimality for data-independent non-uniform or algorithmic-dependent bounds (e.g. uniform stability bounds). The most general notion of optimality we consider here is for data-dependent algorithm-dependent bounds. To explore this notion of optimality, we make the following definitions. To simplify the definition we first introduce a notation for the right hand side of the inequality in Equation (4.3), which we refer to as the *value* of the bound, by letting

$$B(S) = \frac{f_{\mathcal{A}}(S)}{m^\alpha}$$

Definition 4.3.1. A PAC bound of the form in Equation (4.3) with value $B(S)$ is called distribution-admissible or distribution-Pareto optimal for algorithm \mathcal{A} if there does not exist another bound for algorithm \mathcal{A} with value B' such that for all \mathcal{D} , for all $\delta > 0$, and for all m , $\mathbf{E}_{S \sim \mathcal{D}^m}[B'(S)] \leq \mathbf{E}_{S \sim \mathcal{D}^m}[B(S)]$, and $\mathbf{E}_{S \sim \mathcal{D}^m}[B'(S)] < \mathbf{E}_{S \sim \mathcal{D}^m}[B(S)]$ for some \mathcal{D}, δ, m .

Definition 4.3.2. A PAC bound of the form in Equation (4.3) with value $B(S)$ is called optimal with respect to a prior Π over data distributions, for algorithm \mathcal{A} , for a training set size m and confidence level δ if it minimizes the value of $\mathbf{E}_{\mathcal{D} \sim \Pi, S \sim \mathcal{D}^m}[B(S)]$ over valid PAC bounds for algorithm \mathcal{A} .

We also say that the bound is optimal with respect to a prior Π over data distributions, for algorithm \mathcal{A} , if it is optimal in the above sense for all m , and $\delta \in [0, 1]$.

We can extend any of these definitions to require optimality over a family of algorithms rather than a single one. We can also make analogous definitions for uniform bounds. We can then

see that for hypothesis classes for which the Rademacher complexity dominates the $O(1/\sqrt{m})$ term so that the lower bound closely matches the upper bound, then not only is Equation (4.10) (approximately) distribution-Pareto optimal, but it's the unique distribution-Pareto optimal uniform generalization gap bound, and therefore the optimal uniform generalization gap bound for any prior Π (up to lower order terms).

For non-uniform data-dependent bounds, the authors are not aware of any result showing optimality. However, as we will see in the next section, non-uniform bounds are typically based on choosing a “prior”, and the expected value of the bound for different Π depends heavily on the choice of this “prior”. This suggests that perhaps there isn't a unique notion of optimality (i.e. the optimal bound could depend on the assumed true prior Π), but doesn't prove it because these are only upper bounds. We will see in the next section that for some pairs of algorithms and priors, PAC-Bayesian bounds (for binary classification) have a matching lower bound (up to a constant) on the expected value of the generalization error bound and are thus asymptotically optimal according to our definition. This result, combined with the empirical result that the average value of the bound empirically depends on the data distribution, implies that indeed the optimal PAC bound, for a fixed algorithm, depends on the prior over data distributions.

4.4 Marginal-likelihood PAC-Bayesian generalization error bound

In the previous section, we saw that algorithm-independent or data-independent bounds are clearly insufficient to explain the generalization performance of DNNs because the hypothesis class of DNNs is too expressive, and the generalization highly depends on the dataset, respectively. Furthermore, the main approaches for algorithm-dependent bounds are based on non-uniform convergence, which has been shown to have fundamental limitations in its ability to predict generalization in SGD-trained DNNs for some datasets [Nagarajan and Kolter, 2019]. Although there are ways around this limitation (see the discussion in Section 4.2.3.4), it suggests that looking at other approaches to obtain generalization bounds may be promising.

Non-uniform stability bounds offer an interesting alternative to non-uniform convergence, but their empirical success is still limited.

Here we present a new PAC-Bayes bound which applies to a DNN trained using Bayesian inference, with high probability over the posterior. Because it applies to the Bayesian posterior only, it does not apply universally over a large family of posteriors, like standard deterministic PAC-Bayes bounds, which can be shown to sometimes give loose bounds Nagarajan and Kolter [2019]. Furthermore, as we will show in Section 4.5, the bound is asymptotically optimal in the sense defined in Section 4.3.6. One reason why our bound may give significantly tighter results than previous PAC-Bayes bounds applied to DNNs is that our bound works with posteriors and priors in *function space*. Because the parameter-function map [Valle-Pérez et al., 2018] of DNNs is many-to-one, with a lot of parameter-redundancy, it is not hard to construct situations where $KL(Q_{\text{par}}||P_{\text{par}})$ between a parameter-space posterior Q_{par} and prior P_{par} is high, but $KL(Q||P)$ between the induced posterior and prior in function-space is low. In fact, in Section C.7, we show that in general $KL(Q||P) \leq KL(Q_{\text{par}}||P_{\text{par}})$ so that it is always better (or at least not worse) to consider PAC-Bayes bounds in function space for parametrized models, if possible. Furthermore, in Chapter 5, we will empirically verify that our bound gives good predictions for SGD-trained DNNs, and satisfies most of our desiderata for a generalization error bound.

We work in the same set up as Valle-Pérez et al. [2018] and McAllester McAllester [1998]. We consider binary classification, and a space of functions with codomain $\{0, 1\}$, which we call *concepts*. We consider a “prior” P over the concept space \mathcal{H} , and an algorithm which samples concepts according to the Bayesian posterior, with 0 – 1 likelihood. To recall, we define generalization error as the probability of missclassification upon a new sample $\epsilon(c) = \mathbf{P}_{x \sim \mathcal{D}}[c(x) \neq t(x)]$, where t is the target concept. In Section C.5, we prove the following theorem

Theorem 4.4.1. (*Deterministic realizable PAC-Bayes theorem (**marginal-likelihood PAC-Bayes bound**)*) *For any distribution P on any concept space \mathcal{H} and any realizable distribution \mathcal{D} on a space of instances we have, for $0 < \delta \leq 1$, and $0 < \gamma \leq 1$, that with probability at least $1 - \delta$ over the choice of sample S of m instances, that with probability at least $1 - \gamma$ over the choice of c :*

$$-\ln(1 - \epsilon(c)) < \frac{\ln \frac{1}{P(C(S))} + \ln m + \ln \frac{1}{\delta} + \ln \frac{1}{\gamma}}{m-1}$$

where c is chosen according to the posterior distribution $Q(c) = \frac{P(c)}{\sum_{c \in C(S)} P(c)}$, $C(S)$ is the set of concepts in \mathcal{H} consistent with the sample S , and where $P(C(S)) = \sum_{c \in C(S)} P(c)$

The proof is presented in Section C.5. It follows that of the original PAC-Bayesian theorem by McAllister very closely, with the main technical step relying on the quantifier reversal lemma of McAllister McAllester [1998]. Note that the bound is essentially the same as that of Langford and Seeger [2001], except for the fact that it holds in probability and it adds an extra term dependent on the confidence parameter γ , which is usually negligible. The quantity $P(C(S))$ corresponds to the marginal likelihood, or Bayesian evidence of the data S Smith and Le [2017], Germain et al. [2016], and we will also denote it by $P(S)$, to simplify notation. We approximate the marginal likelihood of Bayesian DNNs using the neural network Gaussian process (NNGP) infinite-width approximation [Lee et al., 2017, Matthews et al., 2018, Garriga-Alonso et al., 2018, Novak et al., 2018a, Valle-Pérez et al., 2018] (see Section A.2).

In Valle-Pérez et al. [2018], the authors interpreted $Q(c)$ as approximating the probability by which the stochastic algorithm (e.g. SGD) outputs concept c after training. The preceding bound relaxes this assumption, because it shows that in some sense, the bound holds for “almost all” of the zero-error region of parameter space. More precisely, it holds with high probability over the posterior. This suggests that SGD may not need to approximate the Bayesian inference as closely, for this bound to be useful. Nevertheless, Mingard et al. [2020] give empirical results showing that, for DNNs, the distribution over functions that SGD samples from, approximates the Bayesian posterior rather closely. A fully rigorous generalization error bound for DNNs would need further analysis of SGD dynamics, but we believe these theoretical and empirical results strongly suggest that the PAC-Bayes bound should be applicable to SGD-trained DNNs.

4.5 Bayesian optimality of PAC-Bayesian bound

Now we present a theorem to show that under certain conditions on the algorithm and distribution, the above PAC-Bayes bound is tight. In particular, we will show that if the generalization error decreases as a power law with training set size m , the bound is asymptotically optimal up to a constant.

We again consider binary classification. Let $S_m = \{(x_i, y_i)\}_{i=1}^m$ be a training set of size m .

Consider sampling one more sample (x_{m+1}, y_{m+1}) to obtain $S_{m+1} = \{(x_i, y_i)\}_{i=1}^{m+1}$. We think of S_m and S_{m+1} as random variables with distributions determined by \mathcal{D} . We also define $P_e(x, y, S) := 1 - P(y|S; x)$ to be the probability that a Bayesian posterior P conditioned on a training set S and a test point x predicts the incorrect label. We will denote $P_e(m) := P(x, y, S_m)$, where the dependence on x, y, S_m is left implicit. Note that $\mathbf{E}_{(x,y) \sim \mathcal{D}}[P_e(m)] = \epsilon(\mathcal{A}(S_m))$ where \mathcal{A} is a learning algorithm that returns a function sampled from the Bayesian posterior P conditioned on training set S_m , and ϵ is the average generalization error (averaged both over posterior samples). We will let $\epsilon(m) = \epsilon(\mathcal{A}(S_m))$ where we omit the dependence on \mathcal{A} and S_m for brevity.

In the following we denote with angle brackets $\langle \cdot \rangle$ an average over S_{m+1} (which includes average over S_m). Note that $\langle P_e(m) \rangle = \langle \epsilon(m) \rangle$, and it's also the same as $\epsilon(m)$ averaged over S_m , because $\epsilon(m)$ already includes an average over one test point sampled from \mathcal{D} .

Lemma

all S_{m+1} , $P_e(m) \leq E$, there exists a constant C , such that, as $m \rightarrow \infty$,

$$\langle \log P(S_m) \rangle - \langle \log P(S_{m+1}) \rangle \sim C \langle \epsilon(m) \rangle. \quad (4.21)$$

Furthermore, if $\text{Var}(P_e(m)) = o(\langle \epsilon \rangle)$, then $C = 1$.

The proof is found in Section C.6.

Theorem 4.5.2. *Assume $\langle \epsilon(m) \rangle \sim am^{-\alpha}$ as $m \rightarrow \infty$, with $0 < \alpha < 1$. Then, assuming that for some $E < 1$, for all S_{m+1} , $P_e(m) \leq E$, there exists a constant C' , such that, as $m \rightarrow \infty$,*

$$-\langle \log P(S_m) \rangle \sim C' am^{1-\alpha} \quad (4.22)$$

Furthermore, if $\text{Var}(P_e(m)) = o(\langle \epsilon \rangle)$, then $C' = \frac{1}{1-\alpha}$.

Proof. We can write $-\langle \log P(S_m) \rangle$ using a telescoping sum

$$-\langle \log P(S_m) \rangle = -\langle \log P(S_0) \rangle + \sum_{m'=0}^{m-1} [-\langle \log P(S_{m'+1}) \rangle - -\langle \log P(S'_m) \rangle] \quad (4.23)$$

$$\sim C \sum_{m'=0}^{m-1} \langle \epsilon(m') \rangle \quad (4.24)$$

$$\sim \frac{Ca}{1-\alpha} m^{1-\alpha} \quad (4.25)$$

The second step uses Theorem 4.5.1, and the third uses the assumption in the theorem. Furthermore, from Theorem 4.5.1, the second part of the theorem directly follows. \square

In Theorem 4.5.2, $\langle \epsilon(m) \rangle$ is the actual expected error averaged over training sets of size m . This is a lower bound for the training-set-averaged value of any upper bound on the expected error, like the PAC-Bayes bound [McAllester, 1998]. It is also a lower bound on the average value of high-probability bounds like Section 4.4. On the other hand, the PAC-Bayes bound has an expected value with a leading order behaviour given by

$$\mathbf{E}_{S_m \sim \mathcal{D}^m} [\text{PB}(m)] \sim \frac{-\langle \log P(S_m) \rangle}{m} \sim C'm^{-\alpha}, \quad (4.26)$$

which up to a constant, matches the asymptotic behaviour of the lower bound given by $\langle \epsilon(m) \rangle$ itself. Therefore, we can conclude that the PAC-Bayes bound Section 4.4 is asymptotically optimal (up to a constant, which may be computable a priori), for pairs of priors and learning algorithms that satisfy the theorem assumptions.

The main assumption of the theorem is that $\langle \epsilon(m) \rangle$ follows a power law behaviour asymptotically in m . As we mentioned in Section 4.1, this has been empirically found to be the case for sufficiently expressive deep learning models. Spigler et al. [2019] could prove that the learning curve follows a power law for stationary Gaussian processes in a miss-specified teacher-student scenario, and input instances distributed on a lattice. They could also compute the power law exponents analytically. More recently, Bordelon et al. [2020] developed a theory of average generalization error learning curves (see Section 4.3.5) and proposed an explanation for the power law behaviour based on some assumptions on the data distribution.

The second assumption, which is only necessary to get a smaller value of the constant is

that $\text{Var}(P_e(m)) = o(\langle \epsilon(m) \rangle)$. This seems plausible in many situations. For large m , most of the variation in $P_e(m)$ should typically come from the choice of test point x (on which $P_e(m) = P_e(x, y, S_m)$ implicitly depends) rather than the choice of S_m . We can consider two extreme cases. If $P_e(x, y, S_m)$ is 1 for some x and 0 for all other x , then $P_e(x, y, S_m)$ is a Bernoulli variable, and the variance is of the same order than the mean. On the other extreme, if $P_e(x, y, S_m) = \epsilon(m)$ for all x , then the variance (coming from the choice of x) is 0. It seems plausible that in many situations, we find something in between.

In Chapter 5, we will show extensive empirical evidence that the PAC-Bayes bound follows a power-law behaviour with an exponent closely matching the empirically-measured test error. We sometimes observe deviations, but these are likely coming from the use of the EP approximation, which empirically appears to introduce systematic errors, as a power law [Mingard et al., 2020]. Furthermore, we observe a positive correlation between the exponent α and the proportionality constant relating the bound and the error, C' , as predicted by Theorem 4.5.2.

Chapter 5

PAC-Bayes experiments

In this section we provide experimental evidence that the marginal-likelihood PAC-Bayes bound introduced in Section 4.4 predicts well the change in generalization performance as we change several features of the data and architecture. In particular, we will check that the bound satisfies many of the desiderata proposed in Section 4.1. In order to test the bound against the full set of desiderata, we perform over 1,000 experiments comparing 19 different architectures, ranging from simple fully connected networks (FCNs) and convolutional neural networks (CNNs) to more sophisticated architectures such as Resnet50, Densenet121 and Mobilenetv2. We are thus including variations in architecture hyperparameters such as pooling type, the number of layers, skip connections, etc. . . . These networks are tested on 5 different computer vision datasets of varying complexity. For each of these 95 architecture/data combinations, we train for a range of training set sizes to study learning curves. We additionally study the effect of label corruption on two of the data sets, MNIST and CIFAR-10. As we will see, our bound provides non-vacuous generalization bounds, which are also tight enough to provide a reasonably good quantitative guide to generalization. It also does remarkably well at predicting qualitative trends in the generalization error as a function of data complexity, architecture, and training set size. In particular, we find that we can estimate the value of the learning curve power law exponent for different datasets, which has been the focus of some recent empirical studies [Hestness et al., 2017, Kaplan et al., 2020, Henighan et al., 2020].

In Section 5.1, we will also discuss potential weaknesses of our approach, such as its inability to capture the effect of DNN width or of optimiser choice (but in Chapter 6 we will see that

the latter does not have a big effect on generalization).

To compute the bound of Theorem 4.4.1 for DNNs, we follow the same approach as Valle-Pérez et al. [2018] and use the Gaussian process (GP) approximation to DNNs (also called neural network GP, or NNGP), proposed in several recent works [Lee et al., 2017, Matthews et al., 2018, Novak et al., 2018a, Garriga-Alonso et al., 2018, Yang, 2019b], and which can be used to approximate Bayesian inference in DNNs. This approximation requires computing the GP kernel of the NNGP for the inputs in the training set S , which in the infinite width limit, equals the covariance matrix of the outputs of the DNN with random weights, for inputs in S . This can be computed analytically for FCNs, but for more complex architectures it is unfeasible, so we rely on the Monte Carlo approximation used in Novak et al. [2018a]. We approximate with an estimator of the empirical covariance matrix for the vector of outputs of the DNN, computed from M random initializations of the DNN. The marginal likelihood required to compute Section 4.4 is approximated using the expectation-propagation (EP) approximation, as in Chapter 2 and Chapter 6. See Section B.3 for more details.

In the following experiments, the datasets are binarized, and all DNNs are trained using Adam optimizer to 0 training error. The test error is measured on the full test set for the different datasets used, while the training set is sampled from the full training set provided by these datasets. See the full experimental details in Section A.2.

5.0.1 Error versus label corruption (Desideratum D.1)

Desideratum **D.1** requires that the bound correlates with the error as we change the dataset complexity. We explore this in two ways: by directly corrupting a fraction of the labels of a standard dataset, increasing its complexity, and by comparing different standard datasets with differing complexity. In Figure 5.1, we show the true test error and the PAC-Bayes bound for a CNN, as we increase the label corruption for three different datasets (CIFAR10, MNIST, and Fashion MNIST), which have been binarized. We find that the bound is not only relatively tight, but qualitatively follows the behaviour of the true error, increasing with complexity, as well as preserving the order among the three datasets. In Figure 5.4 and Figure 5.5, we can see more datasets, and can see that for sufficiently large training set size, the bound is typically able to correctly predict in which datasets the networks will generalize better or worse.

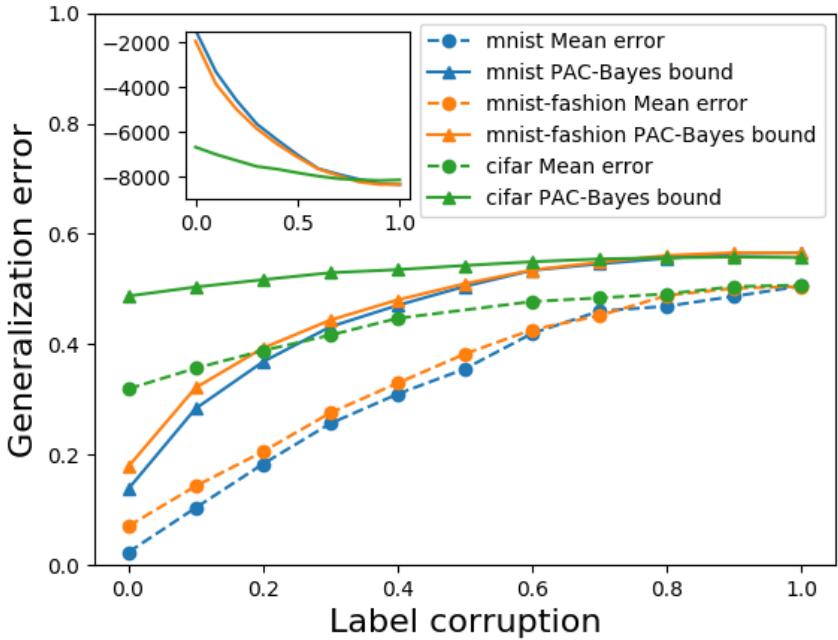


Figure 5.1: **Error with increasing label corruption (D1).** PAC-Bayes bound and generalization error versus label corruption, for a CNN with 4 layers and no pooling, for three datasets and a training set of size 10000. See Section A.2 for more details on architecture. **Insets** show the marginal likelihood $P(S)$ of the data as computed by the Gaussian process approximation (in natural log scale), versus the label corruption.

5.0.2 Error versus training set size (Desideratum D.2)

Desideratum D.2 requires that the bound predicts the change in error as we increase the training set size. Several works [Hestness et al., 2017, Novak et al., 2019b, Rosenfeld et al., 2019, Kaplan et al., 2020] have found that learning curves for DNNs tend to show a power law behaviour with an exponent that depends on the dataset, but not significantly on the architecture¹. We empirically computed the learning curves for many combinations of architectures and datasets, as well as the corresponding PAC-Bayes bounds. The results are shown in Figure 5.4, Figure 5.5, Figure F.1, Figure F.2, and Section F.5. These results agree with the previous empirical work

¹We note that in some work on learning curves for Gaussian processes Sollich [2002], Spigler et al. [2019], Bordelon et al. [2020] more complex types of learning curves have been observed (sometimes involving several regions of non-monotonicity), suggesting that DNNs may potentially show more complex learning curves, different from power laws, in some regimes. As a simple example, in Hestness et al. [2017] it was pointed out that in the law data regime learning curves do not show the asymptotic power law behaviour, which we also see in the double-descent behaviour with respect to data size observed in Nakkiran et al. [2019].

on learning curves, with only a few exceptions, where we observe a deviation from power law behaviour. In particular the learning curve for CIFAR10 for batch 32 seems to deviate from a power law (at least until this value of m). However for batch 256 it shows power law behaviour (see Figure F.9, Figure F.10, Figure F.11 in Section F.5). In Figure 5.2 and Figure 5.4, we show learning curves for three representative architectures for different datasets. Here we clearly see that the exponent of the learning curve depends on the dataset, and that the PAC-Bayes approximately matches the power law exponent of the empirical learning curves, as predicted by Theorem 4.5.2 (see also Figure F.14). We find that, specially for Figure 5.2, the bound even predicts the quick drop in generalization for EMNIST between $m = 4k$ and $m = 15k$ for the FCN. However, we found that this fine grained prediction of the learning curve only held for the FCN. For other architectures, only the overall exponent appears to match, while fluctuations like this don't seem to always agree between the bound and the DNN (see Figure 5.5).

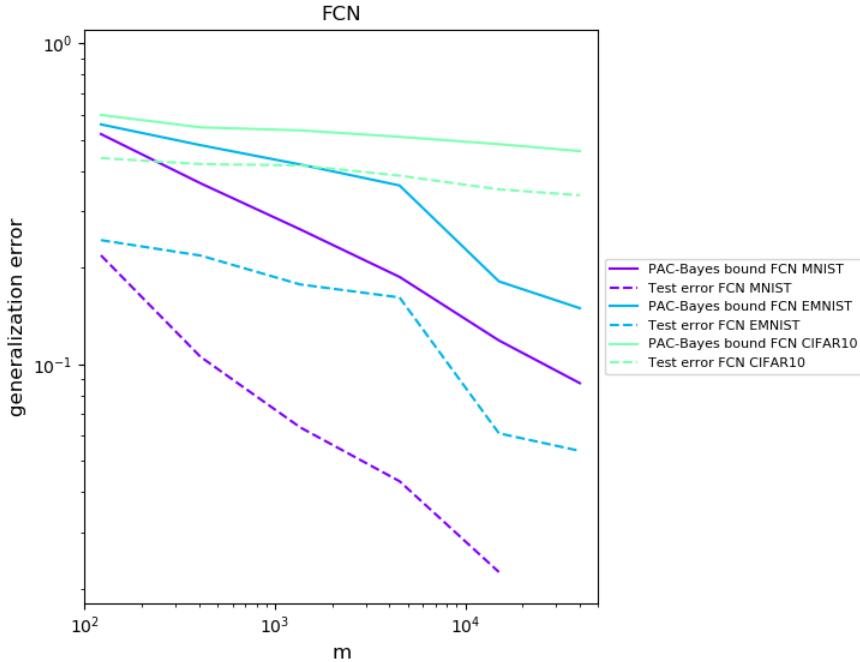


Figure 5.2: Learning curves for fully connected network. Solid and dashed line show, respectively, the empirical test error and the PAC-Bayes bounds for fully connected network (FCN) with 2 hidden layer, versus training set size m , for three different datasets. The DNNs were trained using Adam and batch size 256 to 0 training error.

In Figure F.14, we show the estimated power law exponent of the learning curve, estimated

as a linear fit of $\log \epsilon$ vs $\log m$, versus two ways of estimating the exponent. In Figure F.14a, we estimate the exponent by a linear fit to the log of the PAC-Bayes bound vs $\log m$, and in Figure F.14b, we estimate it as $\alpha = 1 - 1/C'$ where C' is the ratio of the PAC-Bayes bound and the error, which is obtained from the expression in Theorem 4.5.2 when the condition on the variance on error holds. We see that for both ways of estimating the error, there is a good correlation between the estimate of the exponent and the empirical exponent. The absolute value of the estimated exponent does show a more significant deviation from the true value, which is probably due to systematic errors in the EP approximation used to compute the marginal likelihood (this was discussed and empirically investigated in Mingard et al. [2020]), specially in light of the asymptotically optimality of the PAC-Bayes bound (Theorem 4.5.2).

In Figure F.14, we also see that the learning curve exponents cluster according to the dataset, as observed in previous work [Hestness et al., 2017]. However, we also see some smaller, but statistically significant variation in exponents within a dataset, indicating that the architecture does play a role in the learning curve behaviour (though less significant than the dataset). One exception is the FCN which shows a significantly different exponent than other architectures – a deviation which is also predicted by the PAC-Bayes bound.

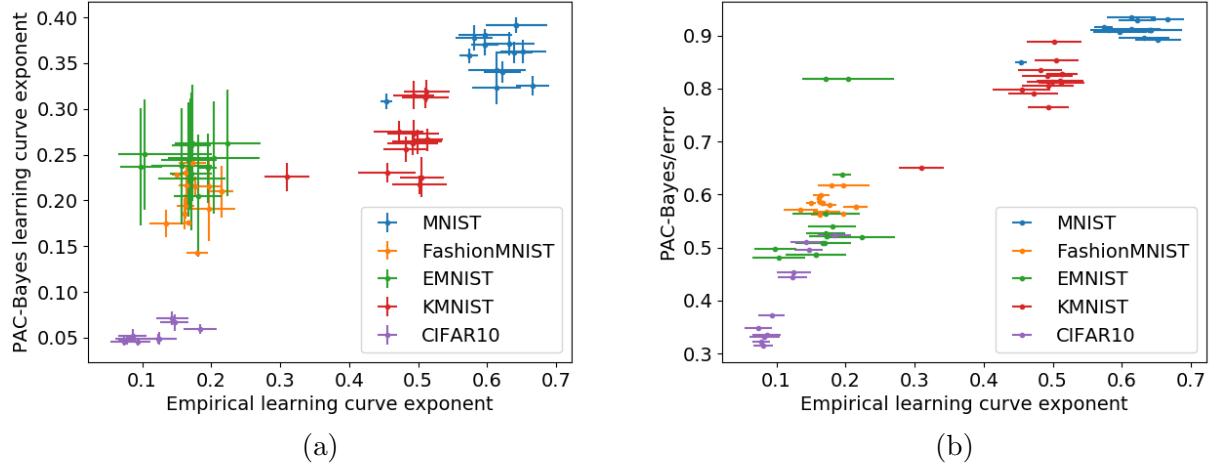


Figure 5.3: **Learning curve exponents of test error vs learning curve exponent estimated from PAC-Bayes bound.** In both plots, each marker shows the learning curve exponent, α obtained from a linear fit to either the learning curve corresponding to $\log \epsilon$ vs $\log m$ or the estimated exponent from the PAC-Bayes bound, for all the architectures and datasets in Section A.2. **(a)** The exponent is estimated from a linear fit to the log of the PAC-Bayes bound vs $\log m$. **(b)** The exponent is estimated as $1 - 1/C'$ where C' is the ratio of the PAC-Bayes bound and the error (see Theorem 4.5.2). The error bars are estimated standard errors from the linear fits. For the ratio estimate the errors due to fluctuations in dataset are negligible. Note that the errors do not take into account errors due to the EP approximation. Note that the exponents cluster according to dataset. The outliers for MNIST and KMNIST are both the FCN. The DNNs were trained using Adam and batch size 32 to 0 training error.

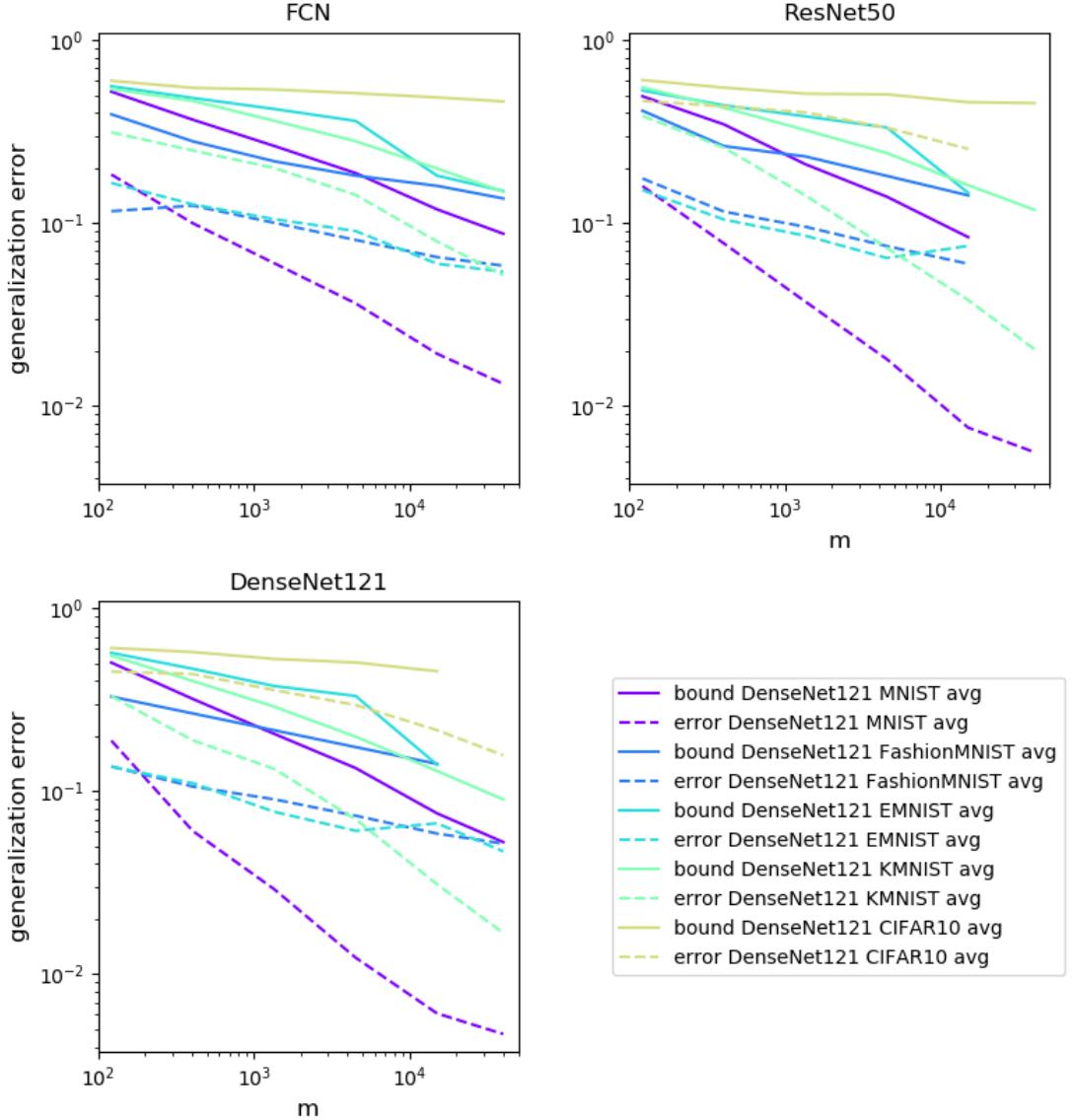


Figure 5.4: **Learning curves for three architectures and different datasets.** Solid and dashed line show, respectively, the empirical test error and the PAC-Bayes bounds. The architectures are a FCN, Resnet50, and Densenet121. The datasets show a range in complexity, ranging from simplest (MNIST) to most complex (CIFAR-10). The DNNs were trained using Adam and batch size 32 to 0 training error.

In Figure 5.5, we show the learning curves for several representative architectures for five different datasets. We see that these different architectures have a very similar learning curve exponent, though there can be a different vertical offset that depends on the architecture. Note

that we made the y-axis range different for the different datasets, to help distinguish the different architectures clearly. In the previous Figure 5.4 and Figure 5.2 one can see the learning curve dependence on dataset more clearly. In Figure F.1 and Figure F.2, in Chapter G, we show the same as in Figure 5.4, but for different variants of resnets and densenets, respectively. We see that within a family of similar architectures, the learning curve is even more similar. For all this range of architectures and datasets we find that the PAC-Bayes bound matches the behaviour of the true error rather closely. In particular, the power law exponent of the PAC-Bayes bound is close to the one of the true learning curve for these 14 different architectures, just as we saw in Figure 5.4 for three representative ones, showing that our generalization error theory is robust and widely applicable. In Figure 5.5, we can even see that the relative ordering of different architectures is typically predicted by the bound, specially for large m . We will see this more clearly in Section 5.0.3.

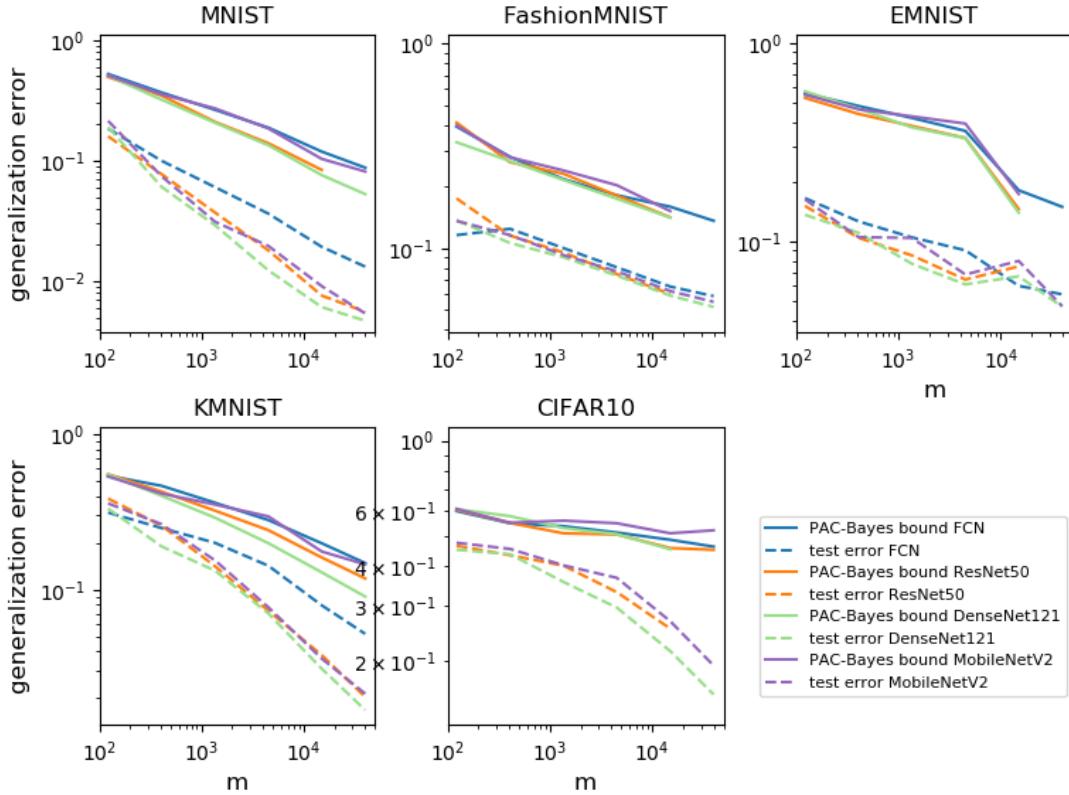


Figure 5.5: **Comparing different architectures.** Learning curves for the test error and the PAC-Bayes bounds for representative architectures and different datasets. Solid and dashed line show, respectively, the empirical test error and the PAC-Bayes bounds. The architectures are a FCN, Resnet50, Densenet121, and MobileNetv2. The DNNs were trained using Adam and batch size 32 to 0 training error. We see the different architectures show similar learning curve power law exponent, which is matched closely by the PAC-Bayes bound. We also see that the PAC-Bayes bound also orders the errors among architectures in a way which agrees well with the true error (Desideratum D.3).

5.0.3 Error versus architecture (Desideratum D.3)

Desideratum D.3 requires that the bound correlates with the error when changing the architecture. We explore this in two ways, by varying certain common architecture hyperparameters (pooling type and depth), and by comparing several SOTA architectures to each other. In Figure 5.6a, we vary the pooling type, and find that the bound correctly predicts that the error is higher for max pooling than avg pooling, and both are higher than no pooling, on this particular dataset. In Figure 5.6b, we vary the number of hidden layers of a CNN trained on MNIST, and find that the bound closely follows the decrease in generalization error.

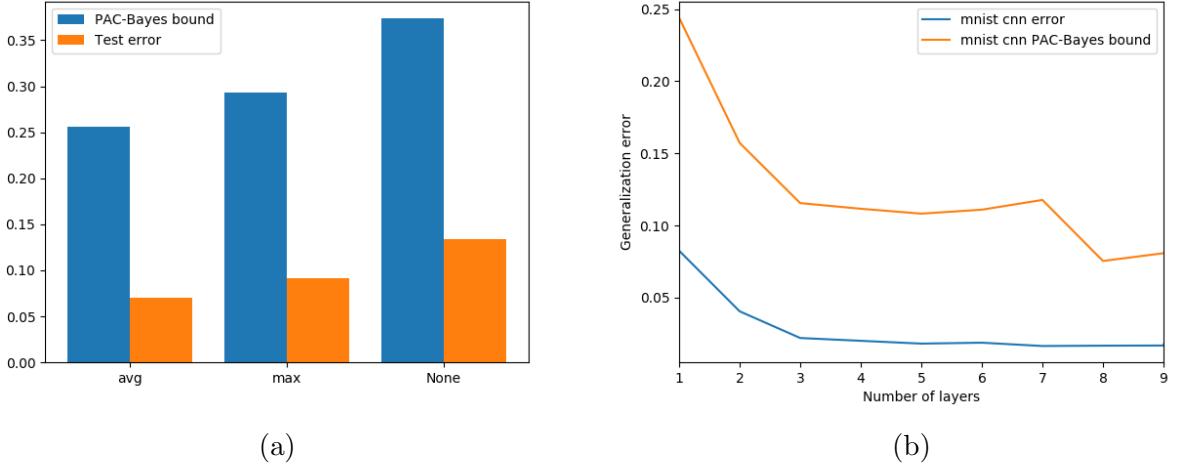


Figure 5.6: **PAC-Bayes bound and generalization error versus different architecture hyperparameters.** (a) error versus pooling type, for a CNN trained on a sample of 1k images from KMNIST. (b) error versus number of layers was for a sample of size 10k from MNIST. Training set error is 0 in all experiments.

To explore more complex changes to the architecture, we plot the bound and error against each other for five datasets, for a set of state-of-the-art architectures, including several resnets and densenet variants (see Section A.2.1 for archiecture details). The results in Figure 5.7 and Figures F.3 to F.6 in Section 5.0.3 display a clear correlation, showing that our PAC-Bayes bound is able to partially explain why some architectures generalize better than others.

In Section F.6 we plot the same results when using batch size 256. The results also show correlation but we have less data is available for those experiments. We plotted the error and bounds for other values of m , and the results are similar, except that the correlation disappears for too small values of m , depending on the dataset. Furthermore, CNNs and FCNs tend to have relatively large values of test error which are sometimes not well captured by the bound, so we chose not to show them on these plots, as to be able to clearly see the variation in test error among the majority of architectures.

Currently, we can't disentangle whether the deviations from the bound predictions are due to deviations from the bound assumptions (for example SGD not behaving as a Bayesian sampler), or from the different approximations used in computing the bound (for example, the EP approximation used in computing the marginal likelihood, see Section B.3).

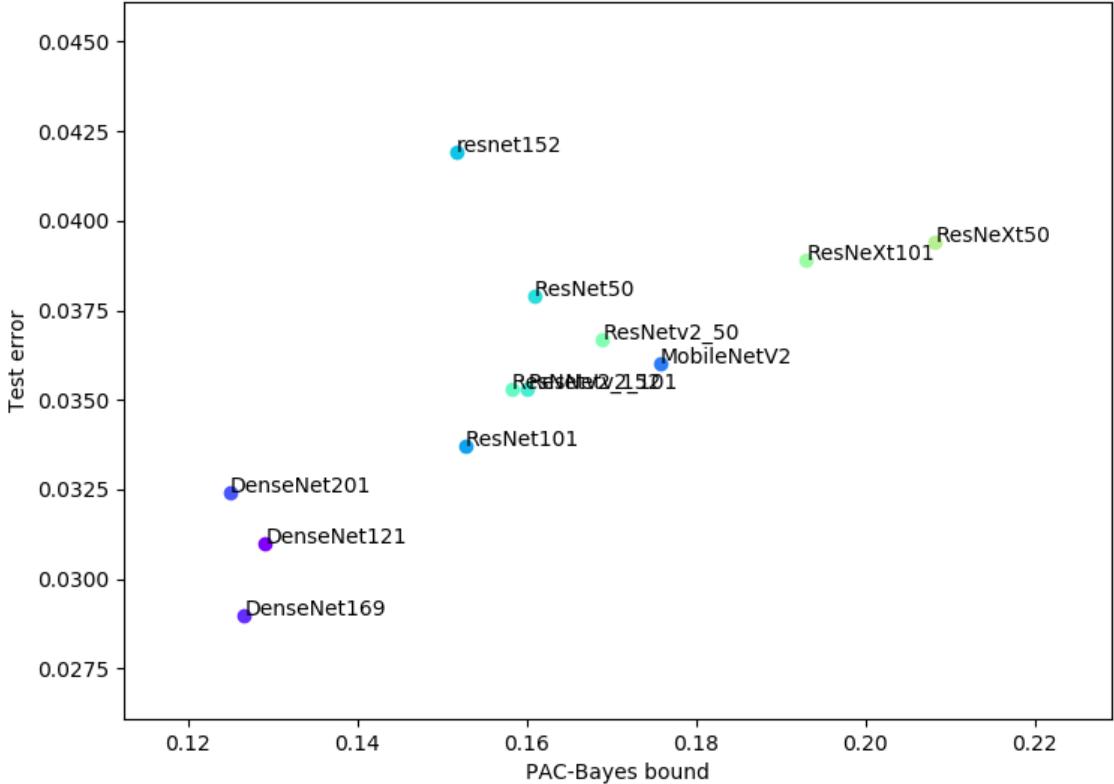


Figure 5.7: PAC-Bayes bound versus test error for different models trained on a sample from KMNIST of size 15k, with batch size 32. FCN and CNNs are removed for clarity as they often have the relatively extreme values of test error and/or bound.

5.1 Discussion of the results

In Chapter 5, we have shown evidence that the PAC-Bayes bound in Theorem 4.4.1, working under the GP approximation, does well at satisfying the desiderata **D.1 -data complexity**, **D.2 - training set size**, and **D.3 - architectures** that we propose. One exception is that our bound, when using the infinite-width GP approximation can't capture the dependence of generalization on layer width (which would fall under desideratum D.3). However, Theorem 4.4.1 might still be able to capture these effects if the marginal likelihood could be computed for finite-width DNNs.

These results also clearly show that **D.5 - non-vacuous** is satisfied. In fact, the logarithm

in the left hand side of Theorem 4.4.1 ensures that the bound is less than 1. Furthermore, in Chapter 5 we find that the bound can be relatively tight as well.

Regarding desideratum **D.7 - rigorous**, Theorem 4.4.1 is fully rigorous for DNNs trained with exact Bayesian inference, in the infinite width limit. However, in Section 4.4 we argue that the theorem is probably applicable to finite-width DNNs trained with SGD (and several of its variants), based on empirical evidence and arguments from Chapter 2 and Chapter 6, as well as the new evidence in the current chapter. But further work is needed to further justify these assumptions.

Regarding desideratum **D.6 - efficiently computable**, our bound lies on the higher end of computation cost among bounds we compare here. The most expensive steps are computing the kernel via Monte Carlo sampling and computing the marginal likelihood. The former has a complexity proportional to $O(m^2)$ times the cost of running the model, but the constant can be reduced by making the last layer wider, and it can be heavily parallelized. On the other hand, computing the marginal likelihood has a complexity of $O(m^3)$ because it requires inverting a matrix, and may only be improved by making assumptions on the kernel matrix (like being low rank). Therefore, the computational complexity of the bound scales similarly to inference in Gaussian processes. This means we could compute up to $m \approx 10^5$, but larger training set sizes would be difficult.

Finally, our approach currently cannot capture effects on generalization caused by different DNN optimization algorithms (**D.7 - training algorithms**), because we rely on the observation that different SGD variants seem to perform similarly, and all seem to approximate Bayesian inference [Mingard et al., 2020]. However, the different algorithms do show some differences in generalization, and capturing that would require an extension to our approach. One of the main hypotheses used to explain differences in generalization among different optimization algorithms is that some algorithms are more biased towards flat solutions in parameter space than others [Hochreiter and Schmidhuber, 1997a, Keskar et al., 2016, Jastrzebski et al., 2018, Wu et al., 2017, Zhang et al., 2018, Wei and Schwab, 2019]. Therefore, one possibility could be to combine our PAC-Bayes approach (based on probabilities of functions), with the more standard approaches based on flatness, to capture this effect.

One of the main results of this work is that our PAC-Bayes bound is able to capture the

power law scaling of the learning curve of DNNs. This has been the subject of recent important work in deep learning theory [Hestness et al., 2017, Kaplan et al., 2020]. However, the work of Kaplan et al. [2020] also highlights the importance of predicting the error as a function of data size, model size (number of parameters) and training iterations, for the purposes of optimizing the loss for a given compute budget. Our theory works at the limit of infinitely large models, trained to convergence (0 training error) so it can only capture the scaling when data is the bottleneck. Extending our theory to finite-sized models and non-zero training error is an important direction for future work.

On the other hand, in a recent follow-up, Henighan et al. [2020], argue the learning curve for the case when very large models are trained to convergence poses a lower bound to the optimal-compute learning curve, which they suggest after a certain compute budget determines the long term evolution of the system. If hypothesis is true, it would mean that the kind of learning curves which we predict in this thesis are the ones that asymptotically determine the fundamental limits of learning with DNN, even when optimal use of compute is taken into account.

Chapter 6

Does SGD approximate Bayesian sampling?

The PF map bias studied in Chapter 2 is a property of untrained networks, with random weights. The PAC-Bayes theorem in Chapter 4 shows that this bias translates to a bias after training that leads to generalization, but the theorem technically applies to DNNs trained with exact Bayesian inference. In practice, DNNs are trained using a variety of stochastic optimization algorithms, of which stochastic gradient descent (SGD) and its variants [Rumelhart et al., 1985, LeCun et al., 2015] are the most popular. In this chapter we will explore what effects the training algorithm has for DNNs, and we will give evidence to show that SGD behaves similarly to Bayesian inference, justifying the use of theories based on Bayesian learning assumptions.

SGD has played a big role in the success of deep learning, due to its effectiveness at training very large DNNs. For example, Zhang et al. [2017a] showed that SGD could find zero training error solutions for a large DNN even when the labels in the dataset were randomized, at the expense of only a moderate increase in training time relative to structured data. Recently, several variants of SGD that can sometimes speed up training have been developed [Schaul et al., 2013, Zeiler, 2012, Kingma and Ba, 2014], but they retain the same structure of changing the weights by a vector proportional to a stochastic estimate of the gradient of the loss. Since the 90s several ideas for improving generalization in SGD-trained DNNs became popular, including penalizing large parameter vectors or ‘weight decay’ [Krogh and Hertz, 1992]; ‘early stopping’, where the training may be stopped before reaching minimum loss to avoid fitting noise [Morgan

and Bourlard, 1990]; ‘dropout’, where neurons are randomly given zero activation value during training, to increase robustness of the learned features [Srivastava et al., 2014]; and ‘batch normalization’ [Ioffe and Szegedy, 2015a], which normalizes the activations of the hidden layers to reduce saturation of the nonlinearities.

Although several of these techniques are still popular for improving generalization, Zhang et al. [2017a] showed that even without using any of these ‘explicit regularization’ techniques, vanilla SGD generalizes well. They find that explicit regularization plays a different role in deep learning than in classical learning models (like linear regression), functioning more as a tuning parameter that can improve generalization, but its absence does not imply poor generalization. Furthermore, the dependence of generalization on hyperparameters tends to be rather complex, which is why large hyperparameter searches often are performed to find good hyperparameter choices for a specific problem.

Zhang et al., therefore consider the question of why do DNNs trained with unregularized SGD generalize. A hope in studying this question is that it may lead us to the fundamental reasons why DNNs generalize, applicable not just to vanilla SGD. Under this perspective, the smaller variations in generalization due to tuning the algorithm or hyperparameters, could be considered ‘second-order’ perturbations on some fundamental mechanism underlying the generalization of DNNs.

Building on classic work on the behaviour of SGD on linear models, Zhang et al. propose that perhaps SGD is implicitly regularizing by biasing the training towards regions of parameter space that correspond to special solutions, for example solutions with low weight norm, that generalize well. This idea has inspired a lot of literature studying properties of SGD that may cause implicit regularization [Neyshabur et al., 2015b, Krueger et al., 2017a, Brutzkus et al., 2018, Zhang et al., 2017b, Poggio et al., 2020].

In this thesis we propose a different approach to the problem of understanding DNN generalization, inspired by the experiments with untrained and Bayesian DNNs in the rest of this thesis. Rather than SGD showing a specific bias in parameter space, we propose that SGD and its variants show a very simple behaviour in parameter space, trying to reach a region of minimum loss, but otherwise not showing significant preference between points in parameter space. This behaviour can be formalized as Bayesian inference with some uninformative prior

in parameter space, like the ones we used in the previous chapters (i.i.d. Gaussian or uniform). If this were true, it would explain why SGD-trained DNNs generalize, and why they also generalize when trained with variants of SGD: the answer would be that all of these training algorithms approximate Bayesian inference and that because of the PF map, the prior in function space is simplicity biased, which leads to generalization in structured problems.

The recent literature studying Bayesian DNNs and NNGPs has proposed that studying Bayesian DNNs could be useful to understand SGD-trained DNNs. For example, a number of papers [Poole et al., 2016, Lee et al., 2018, Valle-Pérez et al., 2018, Yang, 2019a, Mingard et al., 2019, Cohen et al., 2019b, Wilson and Izmailov, 2020] employ arguments on heuristic grounds that the bias in untrained random neural networks could be used to study the inductive bias of optimiser-trained DNNs. Optimiser-trained DNNs have also been empirically compared to their Bayesian counterparts (c.f. Sections 6.6 and 6.7.4 for more detailed discussions). Lee et al. [2017], Matthews et al. [2018], Novak et al. [2018a] used the NNGP approximation to Bayesian DNNs and found that the generalisation performance of Bayesian DNNs and SGD-trained DNNs was relatively similar for standard deep learning datasets such as CIFAR-10, though Wenzel et al. [2020] found more significant differences when using Monte Carlo to approximate finite-width Bayesian DNNs. Others have used either Monte Carlo methods [Mandt et al., 2017] or the GP approximation [Matthews et al., 2017, de G. Matthews et al., 2018, Lee et al., 2019, Wilson and Izmailov, 2020] to examine how similar the Bayesian posterior is to the sampling distribution of SGD (whether in parameter or function space), albeit on relatively low dimensional systems compared to conventional DNNs.

In this chapter we perform extensive computations, for a series of standard DNNs and datasets, of the probability $P_{\text{SGD}}(f|S)$ that a DNN trained with SGD (or one of its variants) to zero error on training set S , converges on a function f . We then compare these results to the Bayesian posterior probability $P_B(f|S)$, for these same functions, conditioned on achieving zero training error on S . In the experiments with GP with MSE loss we also sample from $P_B(f|S)$ so that we could also identify outliers where $P_B(f|S)$ assigns high probability but $P_{\text{SGD}}(f|S)$ assigns low probability, so that they are not found in the sample from $P_{\text{SGD}}(f|S)$. Details on the methods we use are found in Section B.3.

The **main question** we explore here is: *How similar is $P_B(f|S)$ to $P_{\text{SGD}}(f|S)$?* If the two

are significantly different, then we may conclude that SGD provides an important source of inductive bias. If the two are broadly similar over a wide range of architectures, datasets, and optimisers, then, as discussed above, the inductive bias is primarily determined by the prior $P(f)$ of the untrained DNN.

6.1 Main results summary

The work in this chapter was done in collaboration with Chris Mingard et al. Mingard et al. [2020]. The majority of experiments were done by Chris M. co-supervised by Ard A. L and myself. I did the experiments in Section G.2, and the NNGP and neural tangent kernel (NTK) calculations.

We carried out extensive sampling experiments to estimate $P_{\text{SGD}}(f|S)$. Functions are distinguished by the way they classify elements on a test set E . We use the Gaussian process (GP) approximation to estimate $P_B(f|S)$ for the same systems. Our main findings are:

(1) $P_{\text{SGD}}(f|S) \approx P_B(f|S)$ for a range of architectures, including FCNs, CNNs and LSTMs, applied to datasets such as MNIST, Fashion-MNIST, an IMDb movie review database and an ionosphere dataset. This agreement also holds for variants of SGD, including Adam [Kingma and Ba, 2014], Adagrad [Duchi et al., 2011], Adadelta [Zeiler, 2012] and RMSprop [Tieleman and Hinton, 2012].

(2) The $P_B(f|S)$ of functions f that achieve zero-error on the training set S can vary over hundreds of orders of magnitude, with a strong bias towards a set of low generalisation/low complexity functions. This tiny fraction of high probability functions also dominate what is found by DNNs trained with SGD. It is striking that even within this subset of functions, $P_{\text{SGD}}(f|S)$ and $P_B(f|S)$ correlate so well. Our empirical results suggest that, *for DNNs with large bias in $P_B(f|S)$* , SGD behaves *to first order* like a Bayesian optimiser and is therefore exponentially biased towards simple functions with better generalisation. Thus, SGD is not itself the primary source of inductive bias for DNNs.

(3) A function-based picture can also be fruitful for illustrating *second order* effects where an optimiser-trained DNN differs from the Bayesian prediction. For example, training an FCN with different optimisers (OPT) such as Adam, Adagrad, Adadelta and RMSprop on MNIST

generates slight but measurable variations in the distributions of $P_{\text{OPT}}(f|S)$. Such information can be used to analyse differences in performance. For instance, we find that changing batch size affects $P_{\text{Adam}}(f|S)$ but, as was found for generalisation error [Keskar et al., 2016, Goyal et al., 2017, Hoffer et al., 2017, Smith et al., 2017], this effect can be compensated by changes in learning rate. Architecture changes can also be examined in this picture. For example, adding max-pooling to a CNN trained with Adam on Fashion-MNIST increases both $P_B(f|S)$ and $P_{\text{Adam}}(f|S)$ for the lowest-error function f found.

6.2 Preliminaries

Here we summarize the Bayesian formalism we use throughout the experiments in this chapter. See Chapter B for more details.

6.2.1 The Bayesian prior probability, $P(f)$

Given a distribution $P_{\text{par}}(\theta)$ over the parameters, we define the $P(f)$ over functions as

$$P(f) = \int \mathbb{1}[\mathcal{M}(\theta) = f] P_{\text{par}}(\theta) d\theta, \quad (6.1)$$

where $\mathbb{1}$ is an indicator function (1 if its argument is true, and 0 otherwise), and \mathcal{M} is the parameter-function map defined in Section 2.1. This is the probability that the model expresses f upon random sampling of parameters over a parameter initialisation distribution $P_{\text{par}}(\theta)$, which is typically taken to have a simple form such as a (truncated) Gaussian. $P(f)$ can also be interpreted as the probability that the DNN expresses f upon initialisation before an optimisation process. It was shown in [Valle-Pérez et al., 2018] that the exact form of $P_{\text{par}}(\theta)$ (for reasonable choices) does not affect $P(f)$ much (at least for ReLU networks). If we condition on functions that obtain zero generalisation error on a dataset S , then the procedure above can also be used to generate the posterior $P_B(f|S)$ which we describe next.

6.2.2 The Bayesian posterior probability, $P_B(f|S)$

Here, we remind the reader of the Bayesian formalism we use (see also Section B.1), to show how bias in the prior affects the posterior. Consider a supervised learning problem with training data S corresponding to the exact values of the function which we wish to infer (i.e. no noise). This formulation corresponds to a 0-1 *likelihood* $P(S|f)$, indicating whether the data is consistent with the function. Formally, if $S = \{(x_i, y_i)\}_{i=1}^m$ corresponds to the set of training pairs, then we let

$$P(S|f) = \begin{cases} 1 & \text{if } \forall i, f(x_i) = y_i \\ 0 & \text{otherwise.} \end{cases}$$

Note that in our calculations, this quantity is technically $P(S|f; \{x_i\})$, but we denote it as $P(S|f)$ to simplify notation. We will use a similar convention throughout, whereby the input points are (implicitly) conditioned over. We then assume the prior $P(f)$ corresponds to the one defined in Section 6.2.1. Bayesian inference then assigns a *Bayesian posterior probability* $P_B(f|S)$ to each f by conditioning on the data according to Bayes rule

$$P_B(f|S) := \frac{P(S|f)P(f)}{P(S)}, \quad (6.2)$$

where $P(S)$ is also called the *marginal likelihood* or *Bayesian evidence*. It is the total probability of all functions compatible with the training set. For discrete functions, $P(S) = \sum_f P(S|f)P(f) = \sum_{f \in C(S)} P(f)$, with $C(S)$ the set of all functions compatible with the training set. For a fixed training set, all the variation in $P_B(f|S)$ for $f \in C(S)$ comes from the prior $P(f)$ of the untrained network since $P(S)$ is constant. Thus, *the bias in the prior is essentially translated over to the posterior*.

We can also view $P_B(f|S)$ as the distribution over functions that would be obtained by randomly sampling parameters according to $P_{\text{par}}(\theta)$ and selecting only those that are compatible with S .

6.2.3 The optimiser probability, $P_{\text{OPT}}(f|S)$

DNNs are not normally trained by randomly sampling parameters: They are trained by an optimiser. The probability that the optimiser OPT (e.g. SGD) finds a function f with zero error on S can be defined as:

$$P_{\text{OPT}}(f|S) := \int \mathbb{1}[\mathcal{M}(\theta_f) = f] P_{\text{OPT}}(\theta_f|\theta_i, S) \tilde{P}_{\text{par}}(\theta_i) d\theta_i d\theta_f \quad (6.3)$$

where $P_{\text{OPT}}(\theta_t|\theta_i, S)$ denotes the probability that OPT, initialised with parameters θ_i on a DNN, converges to parameters θ_f when training is halted after the first epoch where zero classification error is achieved on S^1 , if such a condition is achieved in a number of iterations less than the maximum number which we allow for the experiments. The initialisation distribution $\tilde{P}_{\text{par}}(\theta_i)$ is defined analogously to $P_{\text{par}}(\theta)$ in Equation (6.1) (though it need not be exactly the same). $P_{\text{OPT}}(f|S)$ is, therefore, a measure of the ‘size’ of f ’s ‘basin of attraction’, which intuitively refers to the set of initial parameters that converge to f upon training.

6.3 Methodology

Here we describe the methodology of the experiments we carry out in this chapter. In Section A.3.1 and Section A.3.2 we give more details on the datasets, and architectures which we use, respectively.

6.3.1 Definition of functions

For a specific DNN, training set $S = \{(x_i, y_i)\}_{i=1}^{|S|}$ and test set $E = \{(x'_i, y'_i)\}_{i=1}^{|E|}$, we define a function f as a labelling of the inputs in S concatenated with the inputs in E .² We will only look at functions which have 0 error on S , so that, for a particular experiment with fixed S and E , the functions are distinguished only by the predictions they make on E . Furthermore

¹In the special case where we specify the experiment as ‘overtraining’, then we take the parameters after p epochs with 0 classification error.

²Formally then, our space of functions is $\mathcal{F} = \mathcal{Y}^{\mathcal{X}}$, where $\mathcal{X} = \{x_i\}_{i=1}^{|S|} \cup \{x'_i\}_{i=1}^{|E|}$

we only consider binary classification tasks (c.f. Section A.3.1), so that our output space³ is $\mathcal{Y} = \{0, 1\}$. Therefore, we will represent functions by a binary string of length $|E|$ representing the labels on E ; the i th character representing the label on the i th input of E , x'_i . For the sake of simplicity, we will not make a distinction between this *representation* of f and the function f itself, as they are related one-to-one for any particular experiment (with fixed S and E).

Restricting the input space where functions are defined can be thought of as a coarse-graining of the functions on the full input space (e.g. the space of all possible images for image classification), which allows us to estimate their probabilities from sampling. In the following subsections we explain how the main experimental quantities are computed. Further detail can be found in Section A.3.

6.3.2 Calculating $P_{\text{OPT}}(f|S)$

For a given optimiser OPT (SGD or one of its variants), a DNN architecture, loss function (cross-entropy (CE) or mean-square error (MSE)), a training set S , and test set E , we repeat the following procedure n times: We sample initial parameters θ_i , from an i.i.d. truncated Gaussian distribution $\tilde{P}_{\text{par}}(\theta_i)$, and train with the optimiser until the first epoch where the network has 100% training classification accuracy (except for experiments labelled “overtraining,” where we halt training after p further epochs with 0 training error have occurred, for some specified p)⁴. We then compute the function f found by evaluating the network on the inputs in E , as described before.

Note that during training, the network outputs are taken to be the pre-activations of the output layer, which are fed to either the MSE loss, or as logits for the CE loss. At evaluation (to compute f), the pre-activations are passed through a threshold function so that positive pre-activations output 1 and non-positive pre-activations output 0.

We chose sample sizes between $n = 10^4$ and $n = 10^6$. In other words, we typically sample over $n = 10^4$ to $n = 10^6$ different trained DNNS, and count how many times each function f appears in the sample to generate the estimates of $P_{\text{OPT}}(f|S)$. We leave the dependence of

³Note that for training with MSE loss, we centered the output so the loss is measured with respect to target values in $\{-1, 1\}$. This is so thresholding can occur at a value of the last layer preactivation of 0, which is the same as for cross-entropy loss on logits.

⁴If SGD fails to achieve 100% accuracy on S in a maximum number of iterations, we discard the run.

$P_{\text{OPT}}(f|S)$ on E implicit. This method of estimating $P_{\text{OPT}}(f|S)$ is described more formally in Section A.3.3.1.

6.3.3 Calculating $P_B(f|S)$ with Gaussian Processes

We use neural network Gaussian processes (NNGPs) [Lee et al., 2017, Matthews et al., 2018, Garriga-Alonso et al., 2019, Novak et al., 2018a] to approximate $P_B(f|S)$, for some training set S and test set E . NNGPs have been shown to accurately approximate the prior over functions $P(f)$ of finite-width Bayesian DNNs [Valle-Pérez et al., 2018, de G. Matthews et al., 2018]. We use DNNs with relatively wide intermediate layers, relative to the input dimension, to ensure that we are close to the infinite layer-width NNGP limit. Depending on the loss function, we estimate the posterior $P_B(f|S)$ as follows:

- **Classification as regression with MSE loss.** As has been done in previous work on NNGPs, we consider the classification labels as regression targets with an MSE loss⁵. We compute the analytical posterior for the NNGP with Gaussian likelihood. This is the posterior over the real-valued outputs at the test points on E , which correspond to the pre-activations of the output layer of the DNN. We sample from this posterior, and threshold the real-values like we do for DNNs (positive becomes 1 and otherwise it becomes 0) to obtain labels on E , and thus a function f . We then **estimate $P_B(f|S)$ by counting how many times each f is obtained from a set of n independent samples from the posterior**, similar to what we did for $P_{\text{OPT}}(f|S)$. For more details on GP computations with MSE loss, see Section B.3.2. We describe this method more formally in Section A.3.3.2. We use this technique when comparing $P_B(f|S)$ with $P_{\text{OPT}}(f|S)$ for MSE loss (e.g. Figure 6.1a).
- **Classification with CE loss.** In several experiments, we approximate the NNGP posterior using a 0-1 misclassification loss, which is more justified for classification, and can be thought of as a “low temperature” version of the CE loss. Since, in contrast to the MSE case, this posterior is not analytically tractable, we use the expectation propagation (EP) approximation to estimate probabilities [Rasmussen, 2004]. In particular, we **estimate**

⁵As for $P_{\text{OPT}}(f|S)$ with MSE loss, we take the regression targets to be $\{-1, 1\}$, so thresholding occurs at 0

$P_B(f|S)$ via ratio of EP-approximated likelihoods.

The EP approximation can be used to estimate the marginal likelihood of any labelling over any set of inputs, given a GP prior. As shown in Equation (6.2), we can use Bayes theorem to express $P_B(f|S)$ as a ratio of $P(f)$ and $P(S)$ (which is valid for functions with 0 error on S , and the 0 – 1 likelihood), and then use EP approximation to obtain both of these probabilities. In the text, when we refer to the EP approximation for calculating $P_B(f|S)$, we are using it as described above.

6.3.4 Comparing $P_{\text{OPT}}(f|S)$ to $P_B(f|S)$

We note that for MSE loss, we can sample to accurately estimate function probabilities, whereas for the CE loss, we must use the EP approximation to calculate the probability of functions.⁶ When we compare $P_B(f|S)$ to $P_{\text{OPT}}(f|S)$ for CE loss, we take the functions found by the optimiser, which are obtained as described in Section 6.3.2, and calculate their $P_B(f|S)$ using the EP approximation. For MSE loss, both the $P_B(f|S)$ and the $P_{\text{OPT}}(f|S)$ are sampled independently, and probabilities are compared for functions found by both methods.

6.3.5 Calculating $P_B(f|S)$ for functions with generalisation error from 0% to 100%

For the zero training error case studied here, we define functions by their particular labelling on the test set E (as described in Section 6.3.1). A function can be generated by picking a certain labelling. Subsequently $P_B(f|S)$ for CE loss can be calculated using the EP approximation as described above. To study how $P_B(f|S)$ varies with generalisation error ϵ_G on E (the fraction of missclassified inputs on E), we perform the following procedure. For each value of ϵ_G chosen, typically 10 functions are uniformly sampled by randomly selecting $\epsilon_G|E|$ bits in E and flipping them. EP is then used to calculate $P_B(f|S)$ for those functions. The probabilities $P_B(f|S)$ can range over many orders of magnitude. The low probability functions cannot be obtained by direct sampling, so that a full comparison with $P_{\text{OPT}}(f|S)$ is not feasible. This is more formally described in Section A.3.3.3.

⁶See Section B.3.1 for more details.

6.3.6 CSR complexity

The critical sample ratio (CSR) is a measure of complexity of functions expressed by DNNs [Krueger et al., 2017b]. It is defined with respect to a sample of inputs as the fraction of those samples which are critical samples. A critical sample is defined to be an input such that there is another input within a box of side $2r$ centred around the input, producing a different output (for discrete classification outputs). See Section G.3 for further details.

6.4 Empirical results for $P_B(f|S)$ v.s. $P_{\text{OPT}}(f|S)$ for different architectures and datasets

In this first of two main results sections, we focus on testing our hypothesis that $P_B(f|S) \approx P_{\text{OPT}}(f|S)$ for FCN, CNN and LSTM architectures on MNIST, Fashion-MNIST, the IMDb review, and the Ionosphere datasets, using several variants of the SGD optimiser. In the following subsection will describe the main results in detail for an FCN on MNIST. The experiments in the next sections will be the same except for the architecture, dataset, or optimiser which will be varied.

6.4.1 Comparing $P_B(f|S)$ to $P_{\text{OPT}}(f|S)$ for FCN on MNIST

In Figure 6.1 we present a series of results for a standard DNN setup: an FCN (2 hidden layers, each 1024 node wide with ReLU activations), trained on (binarised) MNIST to zero training error with a training set size of $|S| = 10,000$ and a test set size of $|E| = 100$. Note that even for this small test set, there are $2^{100} \approx 1.3 \times 10^{30}$ functions with zero error on S , all of which an overparametrized DNN could express [Zhang et al., 2017a]⁷. We chose standard values for batch size, learning rate, etc., if given by the default values in Keras 2.3.0 (e.g. batch size of 32 and learning rate of 0.01 for SGD). Our experiments in Section 6.5 and the appendices will show that our results are robust to the choice of these hyperparameters.

Figure 6.1a compares the value of $P_B(f|S)$ and $P_{\text{SGD}}(f|S)$ for the highest probability functions

⁷We also find in Figure G.15a and Figure G.15b that our 2-layer FCN is capable of expressing functions on MNIST with the full range of training and generalisation errors

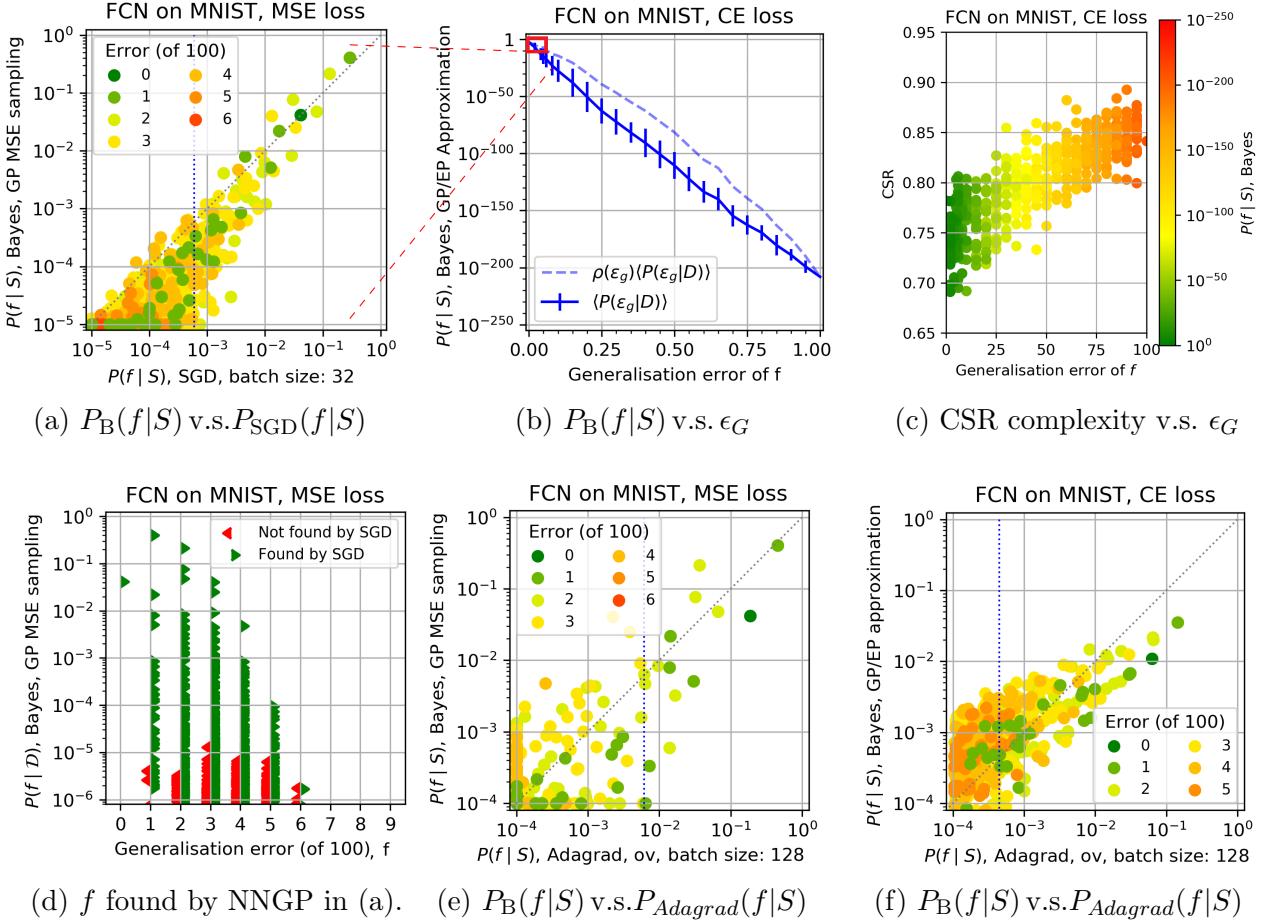


Figure 6.1: Comparing the Bayesian prediction $P_B(f|S)$ to $P_{OPT}(f|S)$ for SGD and Adagrad, for an FCN on MNIST [We use training/test set size of 10,000/100; For (a,e,f), the vertical dotted blue lines are drawn at the highest value of $P_{OPT}(f|S)$ such that the sum of $P_{OPT}(f|S)$ for all functions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

(a) $P_B(f|S)$ v.s. $P_{SGD}(f|S)$ for MSE loss; Both $P_B(f|S)$ and $P_{SGD}(f|S)$ were sampled $n = 10^6$ times. The color shows the number of errors in the test set. The GP has average error $\langle \epsilon_G \rangle_{GP} = 1.61\%$, while SGD has average error $\langle \epsilon_G \rangle = 1.88\%$.

(b) $P_B(f|S)$ (with CE loss) v.s. ϵ_G for the full range of possible errors on E . We use the methods from Section 6.3.5 with 20 random functions sampled per value of error. The solid blue line shows $\langle \log(P_B(f|S)) \rangle_{\epsilon_G}$, where the average is over the functions for a fixed ϵ_G ; error bars are 2 standard deviations. The dashed blue line shows the weighted $\rho(\epsilon_G) \langle P_B(f|S) \rangle_{\epsilon_G}$, where $\rho(\epsilon_G)$ is the number of functions with error ϵ_G . The small red box and dashed red lines illustrate the range of probability and error found in (a).

(c) CSR complexity versus generalisation error for the same functions as in fig (b). Color represents $P_B(f|S)$, computed as in (b).

(d) Functions from (a) found by the sample of $P_B(f|S)$, versus error. 913 functions of the functions are also found by SGD, taking up 97.70% of the probability for $P_{SGD}(f|S)$, and 99.96% for $P_B(f|S)$.

(e) $P_B(f|S)$ v.s. $P_{Adagrad}(f|S)$ for MSE loss; $P_{Adagrad}(f|S)$ was sampled $n = 10^5$ times (while the GP sample was the same as in (a)). Adagrad was overtrained until 64 epochs had passed with zero error. The average error is $\langle \epsilon_G \rangle = 1.53\%$.

(f) is as (e) but with CE loss, so that the EP approximation was used for $P_B(f|S)$, making the estimate of $P_B(f|S)$ slightly less accurate. $\langle \epsilon_G \rangle = 2.63\%$.

of each distribution, for MSE loss. Each data point in the plot corresponds to a unique function (a unique classification of images in the test set E). The functions are obtained by sampling both $P_B(f|S)$ and $P_{SGD}(f|S)$ and taking the union of the set of functions obtained. $P_B(f|S)$ and $P_{SGD}(f|S)$ were estimated as frequencies from the corresponding sample as explained in Sections 6.3.2 and 6.3.3. If a function does not appear in one of the samples, we set its frequency to take the minimum value so that it would appear on top of one of the axes. For example, a function that appears in the SGD runs, but not in the sampling for $P_B(f|S)$, will appear on x-axis at the value obtained for $P_{SGD}(f|S)$. Here we used MSE loss rather than the more popular (and typically more computationally efficient) CE loss because for MSE, the posterior can be sampled from without further approximations, while for CE loss, the expectation propagation (EP) approximation needs to be used making $P_B(f|S)$ less accurate (see Chapter B for further details).

Figure 6.1a also demonstrates that $P_{SGD}(f|S)$ and $P_B(f|S)$ are remarkably closely correlated for MSE loss, and that a remarkably small number of functions account for most of the probability mass for both $P_B(f|S)$ and $P_{SGD}(f|S)$. To appreciate how remarkably tight this agreement is, consider the full scale of probabilities for functions f that achieve zero error on the MNIST training set. The average $P_B(f|S)$ of all these functions is $2^{-100} \approx 10^{-30}$. Therefore the functions in Figure 6.1a have probabilities that are many orders of magnitude higher than average. At the same time, $P_B(f|S)$ and $P_{SGD}(f|S)$ for these functions typically agree within less than one order of magnitude. Another way of quantifying the agreement is that 90% of the cumulative probability weight from both $P_{SGD}(f|S)$ and $P_B(f|S)$ for the test set E in Figure 6.1a is made up from the contributions of only a few tens of functions with zero training error out of $\approx 10^{30}$ such possible functions (see vertical dotted line in Figure 6.1a). Moreover, these particular functions are the same for both $P_B(f|S)$ and $P_{SGD}(f|S)$. The agreement between the two methods is remarkable. Overall, the observations in Figure 6.1a suggest that the main inductive bias of this DNN is present prior to training.

Figure 6.1b plots the mean probability for obtaining a generalisation error of ϵ_G in the training set E , which is estimated as $\rho(\epsilon_G)\langle P_B(f|S) \rangle_{\epsilon_G}$ where $\rho(\epsilon_G) = |E|!/((|E| - \epsilon_G|E|)!(\epsilon_G|E|)!)$ denotes the number of functions with $\epsilon_G|E|$ errors on E , and $\langle P_B(f|S) \rangle_{\epsilon_G}$ denotes the expected value of $P_B(f|S)$, where the expectation is with respect to uniformly sampling from the set

of functions with fixed ϵ_G . As explained in Section 6.3.5, we estimate the average $\langle \cdot \rangle_{\epsilon_G}$ by sampling, and we estimated $P_B(f|S)$ for each f in the sample using the EP approximation.

Figure 6.1b can be interpreted as showing that the inductive bias encoded in $P_B(f|S)$ gives good generalisation. More precisely, we find that $P_B(f|S)$ is exponentially biased towards functions with low generalisation error. To illustrate how strong the bias is, we can look at $\rho(\epsilon_G)$. Over 50% of functions are in the range of $\epsilon_G = 50 \pm 3$ errors, while only $10^{-23}\%$ have $\epsilon_G \leq 3$. Therefore for $P_B(f|S)$ to overcome the ‘entropic’ factor $\rho(\epsilon_G)$ and show the behaviour in Figure 6.1b, it must in average give a probability many orders of magnitude higher to low error functions than to high error functions. In Section A.3.3.3, we also observed that the probability p_i of misclassifying an image in the test set varies a lot between images, and that these probabilities are to first order independent. As a corollary, in Figure G.16 we show for $P_B(f|S)$ and $P_{SGD}(f|S)$ that the probabilities of multiple images being misclassified can be accurately estimated from the products of the probabilities p_i for misclassifying individual images. Thus this system appears to behave like a Poisson-Binomial distribution with independent and non-identically distributed random p_i , which most likely also explains why $\langle \log P_B(f|S) \rangle_{\epsilon_G}$ scales nearly linearly with ϵ_G .

Although we cannot measure $P_{SGD}(f|S)$ for the high generalisation error functions, the agreement in Figure 6.1a (and elsewhere in this chapter) implies that $P_{SGD}(f|S)$ must also be on average orders of magnitude lower for high error functions than low error functions. However, at the moment we can only conjecture that $P_{SGD}(f|S)$ follows the same exponential behaviour as $P_B(f|S)$ over the whole range of ϵ_G . Finally, in Section A.3.3.3, we make some further remarks and caveats about this experiment, and other similar experiments.

Figure 6.1c shows the correlation between the complexity of the functions obtained to create Figure 6.1b, and their generalisation error, as well as their $P_B(f|S)$ (from EP approximation) represented in their color. The complexity measure we used is the critical sample ratio (CSR) complexity [Krueger et al., 2017b] computed on the inputs in E , which measures what fraction of inputs are near the decision boundary (see Section 6.3.6).

Figure 6.1c also shows that there is a inverse correlation between the generalisation of a function and its CSR complexity, as well as between $P_B(f|S)$ and CSR. In Section 6.2.2, we showed that $P_B(f|S)$ is proportional to the prior probability of a function $P(f)$ for functions

that have zero error on the training set S . We can thus understand the inverse correlation between $P_B(f|S)$ and CSR in the light of previous *simplicity bias* results showing that the prior $P(f)$ of Bayesian DNNs is exponentially biased towards functions with low Kolmogorov complexity (simple functions) [Valle-Pérez et al., 2018, Mingard et al., 2019]. In [Valle-Pérez et al., 2018], it was further shown for an FCN on a subsample of MNIST that $P(f)$ correlated remarkably well with CSR⁸, and our results are in agreement with that finding. The results in this figure extend those of Figure 6.1b to show that $P_B(f|S)$ is biased both towards low error and simple functions, and that simple functions are the ones that tend to have good generalisation on MNIST.

Figure 6.1d shows the correlation between $P_B(f|S)$ and ϵ_G for functions used for Figure 6.1a. We note that, as can also be observed in Figure 6.1a, values of $P_B(f|S)$ are high for low error functions, and high error functions have relatively lower values of $P_B(f|S)$. This figure also uses colour to show which functions were not found in the sampling of $P_{SGD}(f|S)$. It shows clearly that SGD finds all the high $P_B(f|S)$ functions.

Figure 6.1e shows the same type of experiment as in Figure 6.1b, but using a different SGD-based optimiser, Adagrad [Duchi et al., 2011] with overtraining (where training was halted after 64 epochs had passed with 100% training accuracy). We see that it exhibits similar correlation between $P_B(f|S)$ and $P_{OPT}(f|S)$ to vanilla SGD (and very similar agreement was observed without overtraining). We will see throughout the chapter remarkably good correlations between $P_B(f|S)$ and $P_{OPT}(f|S)$ holds for a large range of optimisers and hyperparameters

Figure 6.1f shows the same type of experiment as in Figure 6.1a, but using CE loss, the Adagrad optimiser, and overtraining (also to 64 epochs). See Figure G.10b for the equivalent plot but without overtraining. As we are using CE loss (see Section 6.3.3 and Section 6.3.4), we sample functions from $P_{OPT}(f|S)$, and then use the EP to estimate $P_B(f|S)$ for the functions obtained. We find similar results to Figure 6.1e, where we used MSE loss (and direct sampling for $P_B(f|S)$). The errors introduced by the EP approximation may explain why the correlation does not follow the $x=y$ line as closely as it does for the MSE calculations. Nevertheless, the correlation between $P_B(f|S)$ and $P_{Adagrad}(f|S)$ is strong, providing evidence that our results

⁸Furthermore in [Valle-Pérez et al., 2018], it was shown that this was not an exclusive property of CSR and that any measure that could approximate Kolmogorov complexity seems to also correlate well with $P(f)$.

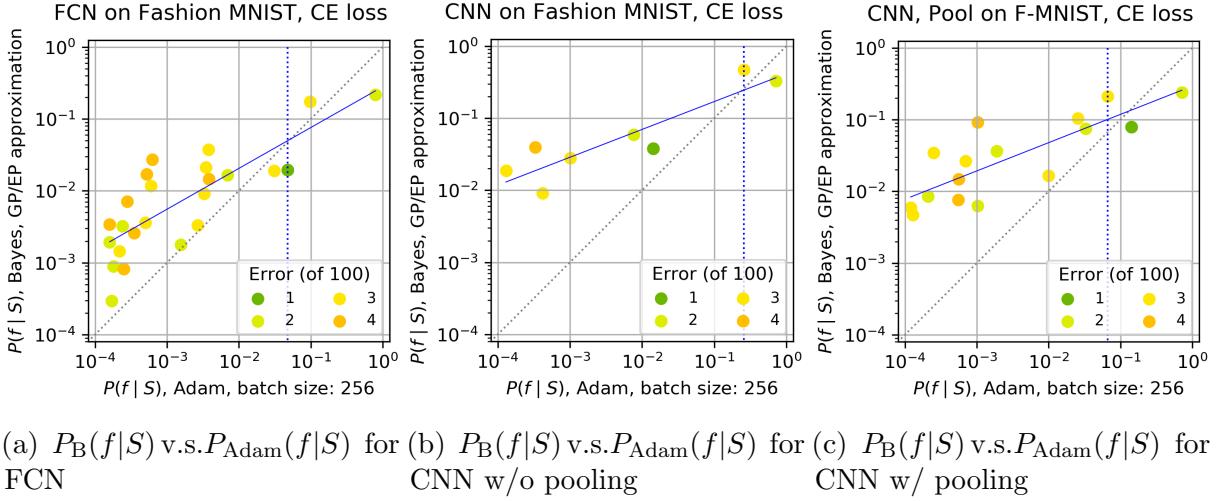


Figure 6.2: Comparing $P_B(f|S)$ to $P_{\text{Adam}}(f|S)$ for CNNs and the FCN on Fashion-MNIST [We use a training/test set size of 10,000/100; vertical dotted blue lines denote 90% probability boundary; dashed grey line is $P_B(f|S) = P_{\text{OPT}}(f|S)$.] **(a)** FCN on Fashion-MNIST; $\langle \epsilon_G \rangle = 2.11\%$ for Adam with CE loss. **(b)** Vanilla CNN on Fashion-MNIST; $\langle \epsilon_G \rangle = 2.25\%$ for Adam with CE loss. **(c)** CNN with max-pooling on Fashion-MNIST; $\langle \epsilon_G \rangle = 1.96\%$ for Adam with CE loss. Note that when max-pooling is added, the probability of the lowest-error function increases notably for both $P_{\text{Adam}}(f|S)$ and $P_B(f|S)$. There is a strong correlation between $P_B(f|S)$ and $P_{\text{SGD}}(f|S)$ in all three plots. See Figure G.3 for related results, including $P_B(f|S)$ vs ϵ_G , a CNN with batch normalisation, and a CNN with MSE loss.

for an FCN on MNIST are not an artefact of the exact optimiser or loss function used.

6.4.2 Comparing $P_B(f|S)$ to $P_{\text{Adam}}(f|S)$ for CNNs on Fashion-MNIST

We next turn to a more complex dataset, namely Fashion-MNIST [Xiao et al., 2017] which consists of images of clothing, as well as a more complex network architecture, the CNN [LeCun et al., 1999] which was designed in part to have a better inductive bias for images. See Section A.3.1 and Section A.3.2 for details on dataset and architecture. We can see in Figure 6.2 a strong correlation between $P_B(f|S)$ and the probabilities found by the Adam optimiser [Kingma and Ba, 2014], a variant of SGD. Note that instead of MSE loss we used CE loss because it is more efficient. A downside of this choice is that we need to use the EP approximation for the GP calculations (see Section B.3.1). Although the correlation is strong, it does not follow $x=y$ as closely as we generally find for MSE loss, which is quite possibly an effect of the EP approximation. See Figure G.3 for an example with MSE loss where the

correlation does follow $x=y$ more closely. Both the FCN and the CNNs exhibit a strong bias towards low error functions on Fashion-MNIST as we can see in Figure G.3c and Figure G.3d.

For an example of how the effects of architecture modifications can be observed in the function probabilities, compare results in Figure 6.2b for the vanilla CNN to those in Figure 6.2c for a CNN with max-pooling [He et al., 2016a], a method designed to improve the inductive bias of the CNN. As expected, the generalisation performance of the CNN improves, and an important contributor is the increase in the probability of the highest probability 1-error function in both $P_B(f|S)$ and $P_{\text{Adam}}(f|S)$, directly demonstrating an enhancement of the inductive bias. See Figure G.3 for related results. This example demonstrates how a function based picture as well as analysis of the Bayesian $P_B(f|S)$ sheds light on the inductive bias of a DNN. Such insights could help with architecture search, or more generally with developing new architectures with improved implicit bias toward desired low error functions.

6.4.3 Comparing $P_B(f|S)$ and $P_{\text{SGD}}(f|S)$ to Neural Tangent Kernel results

In Figure 6.3 we compare $P_B(f|S)$ to the output of the neural tangent kernel (NTK) [Jacot et al., 2018], which approximates gradient descent in the limit of infinite width and infinitesimal learning rate. The generalisation error of NTK and NNGPs have been shown to be relatively close, and they produce similar functions on simple 1D regression [Lee et al., 2019, Novak et al., 2019b]. Here we show that this similarity also holds for the function probabilities for a more complex classification task. However, we also find the NTK misses many relatively high probability functions that both SGD and the GP find. We are currently investigating this surprising behaviour, which may arise from the infinitesimal learning rate. Their low probability may also be exacerbated by the fact that in Figure 6.3 the NTK is very highly biased towards one 2-error function, forcing other functions to have low cumulative probability. Again, this example demonstrates how a function based picture picks up rich details of the behaviour that would be missed when simply comparing generalisation error.

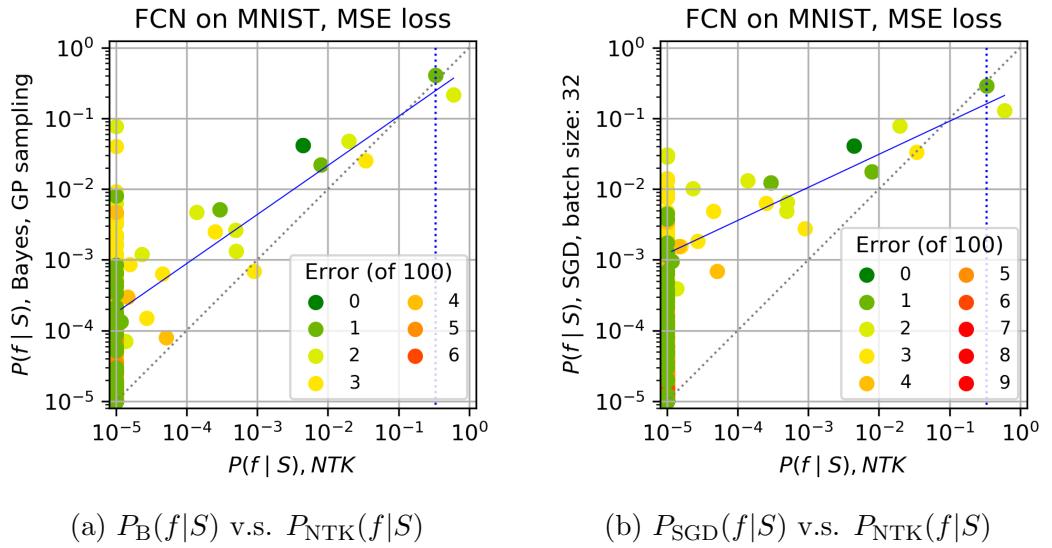


Figure 6.3: **Comparing $P_B(f|S)$ and $P_{SGD}(f|S)$ to $P_{NTK}(f|S)$ for an FCN on MNIST.** [The functions to the right of the blue dotted lines make up 90% of the total probability. We did 10^7 samples for NTK and GP, and 10^6 for SGD]. In (a) we show the correlation between $P_{NTK}(f|\mathcal{D})$ and $P_B(f|S)$. Weighted by probability, 77.5% of functions found by sampling from the GP are found by NTK; all functions found by NTK are found by sampling from the GP. In (b), we show the correlation between the $P_{NTK}(f|\mathcal{D})$ and $P_{SGD}(f|S)$. Weighted by probability, 65.8% of functions found by SGD are found by NTK; all functions found by NTK are found by SGD. $\langle \epsilon_G \rangle = 1.69\%$ (NTK), $\langle \epsilon_G \rangle = 1.61\%$ (GP), $\langle \epsilon_G \rangle = 1.88\%$ (SGD).

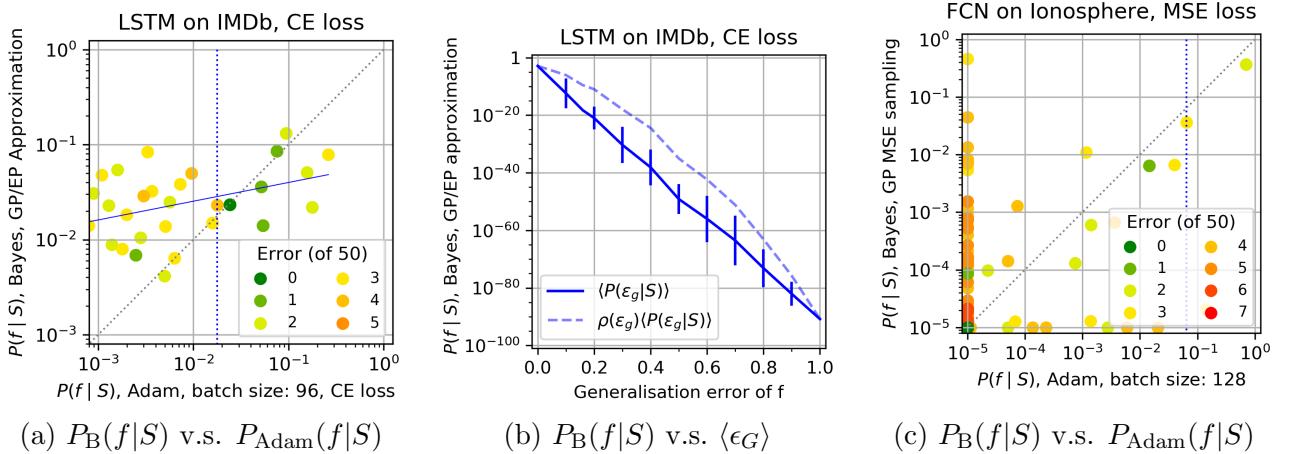


Figure 6.4: **Comparing $P_B(f|S)$ to $P_{\text{Adam}}(f|S)$ for a LSTM on the IMDb movie review dataset, and an FCN on the ionosphere dataset.** (a) $P_B(f|S)$ v.s. $P_{\text{Adam}}(f|S)$ for LSTM on IMDb dataset, ($\langle \epsilon_G \rangle = 4.28\%$, 10^4 samples). Because of the computational cost of the problem, we used a training set size of 45000 and a test set of size 50. (b) $P_B(f|S)$ v.s. $\langle \epsilon_G \rangle$ for the LSTM on IMDb shows that the functions found by the Adam optimiser are in the small fraction of high $P_B(f|S)$ probability/low error functions. (c) $P_B(f|S)$ v.s. $P_{\text{Adam}}(f|S)$ for an FCN with 3 hidden layers of width 256 on the Ionosphere dataset. Training set size is 301 and the test set size is 50. ($\langle \epsilon_G \rangle = 4.59\%$ for Adam, $\langle \epsilon_G \rangle = 5.41\%$ for the GP). See Figures G.4 and G.5 for further results for these systems.

6.4.4 Comparing $P_B(f|S)$ to $P_{\text{Adam}}(f|S)$ for LSTM on IMDb sentiment analysis

We test a more complex DNN with a LSTM layer [Hochreiter and Schmidhuber, 1997b], applied to a problem of sentiment analysis on the IMDb movie database. We used a smaller test set $|E| = 50$ and a larger training set $|S| = 45,000$ to ensure that generalisation was good enough to ensure that functions are found with sufficient frequency to be able to extract probabilities. As can be seen in Figure 6.4a we again observe a reasonable correlation between the functions found by Bayesian sampling, and those found by the optimiser. Figure 6.4b also shows that, as observed for other datasets, this system is highly biased towards low error functions. We show some further experiments with the LSTM in Figure G.4 in Section G.1, including an experiment with MSE loss to avoid the EP approximation.

6.4.5 Comparing $P_B(f|S)$ to $P_{\text{Adam}}(f|S)$ for FCN on Ionosphere dataset

As another non-image classification example, we use the small non-image Ionosphere dataset (with a training set of size 301), using an FCN with 3 hidden layers of width 256. As can be seen in Figure 6.4c, for MSE loss we find a fairly good correlation. Further details and an example with CE loss can be found in Figure G.5.

6.4.6 Effects of training set size

We performed experiments comparing $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$ for different training set sizes for the FCN on MINST. We observe that increasing the amount of training data from $|S| = 1000$ to $|S| = 20000$ increases the bias towards low error functions. This increase has the following effects: 1) An increase in the value of $P_B(f|S)$ and $P_{\text{SGD}}(f|S)$ for functions with low $\langle \epsilon_G \rangle$ by several orders of magnitude, 2) an increase by several orders of magnitude of $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$ for the mode functions (the ones with highest probability), 3) A decrease in the number of functions that cumulatively take up 90% of the observed probability weight, and 4) a significant increase in the tightness of correlation between $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$. See Section G.1.2 in Section G.1.2 for detailed results and plots.

6.4.7 Results for other test sets

For the experiments shown in this section, sampling efficiency considerations means that we have limited ourselves to a relatively small test sets ($|E| \leq 100$). We have checked that other test sets also show close agreement between $P_B(f|S)$ and $P_{\text{SGD}}(f|S)$ (see e.g. Figure G.1). For larger $|E|$, $P_{\text{SGD}}(f|S)$ quickly becomes impossible to directly measure empirically (doubling the test set roughly means squaring the number of samples to obtain qualitatively similar results). But a larger set can be seen as the concatenation of smaller sets. If we assume that the images are approximately independently distributed, as Section G.4 suggests, then we can estimate the highest probabilities from products of $P_B(f|S)$ or $P_{\text{SGD}}(f|S)$ on the smaller sets.

6.5 The effect of hyperparameter changes and optimisers on $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$

In the first section we focussed on the first-order similarity between $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$. In this second main results section, we focus on second-order effects that affect $P_{\text{OPT}}(f|S)$ differently from $P_B(f|S)$. These include the effects of hyperparameter settings and optimiser choice.

6.5.1 Changing batch size and learning rate

In a well-known study, [Keskar et al., 2016] showed that, for a fixed learning rate, using smaller batch sizes could lead to better generalisation. In Figure 6.5 (a)-(c) we observe this same effect but reflected in the more finely grained spectrum of function probabilities. For batch size 512, we also reproduce in Figure 6.5d the effect observed in [Goyal et al., 2017, Hoffer et al., 2017, Smith et al., 2017], that speeding up the learning rate for a fixed batch size can mimic the improvement in $\langle \epsilon_G \rangle$ for smaller batches. Interestingly, as can be seen by comparing Figures 6.5d to 6.5f, the overall correlation of the function probability spectrum appears tighter for the 128 and 512 batch size with the same learning rates, even though the generalisation errors are different. However, if the learning rate is increased 4× for the the 512 batch size system, then there is a closer correlation with batch size 128 for the higher probability functions. It is these latter functions that dominate the average for $\langle \epsilon_G \rangle$ and so the closer correlation for those functions, rather than the less good correlation for low probability functions, explains the better agreement seen in generalisation error for the two systems.

Finally, in Figure G.8 of Section G.1, we vary batch size for MSE, finding different trends to CE loss. For MSE, increasing batch size leads to better generalisation due to second order effects where $P_{\text{SGD}}(f|S)$ preferentially converges on a few key higher probability/lower error functions. The batch size can be correlated with the noise spectrum of the underlying Langevin equation that describes SGD [Bottou et al., 2018, Jastrzebski et al., 2018, Zhang et al., 2018]. What our function based results demonstrate is that the behaviour of the optimiser on the loss-landscape is affected in subtle ways by the form of the loss function, as well as the amount

noise, and possibly also by correlations in the noise.

6.5.2 Changing optimisers

We trained the FCN on MNIST with different optimisers (Adam, Adagrad, RMSprop, Adadelta), and found that to first order $P_B(f|S)$ correlated well with $P_{\text{OPT}}(f|S)$ for all four optimisers. We also observed some second order effects, including that the distribution of $P_{\text{Adam}}(f|S)$ and $P_{\text{Adagrad}}(f|S)$ were very similar to one another, as were $P_{\text{RMSprop}}(f|S)$ and $P_{\text{Adadelta}}(f|S)$, but there was more noticeable variation between the two groups. We find that $P_{\text{Adam}}(f|S)$ with batch size of 32 is very similar to $P_{\text{RMSprop}}(f|S)$ with a batch size of 128. The effect of optimiser choice, batch size, learning rate, and other hyperparameters is complex, and the hyperparameter space is large. Analysing optimisers in function-space could be a way to better understand the interaction of these choices with the loss landscape, and understanding the effects of hyperparameter tuning. See Section G.1.1 for further detail on the experiment results and the plots.

6.6 Heuristic arguments for the correlation between $P_B(f|S)$ and $P_{\text{SGD}}(f|S)$

At first sight it may seem rather surprising that SGD, which follows gradients down a complex loss-landscape, should converge on a function f with a probability anything like the Bayesian posterior $P_B(f|S)$ that upon random sampling of parameters, a DNN generates functions f conditioned on S . Indeed, in the general case of an arbitrary learner we don't expect this correspondence to hold. However, as shown e.g. in Fig 1, $P_B(f|S)$ is orders of magnitude larger for functions with small generalisation error than it is for functions with poor generalisation. As explained in Sections 6.7.3 and 6.7.5, such an exponential bias towards low complexity/low error functions can be expected on fairly general grounds [Valle-Pérez et al., 2018, Mingard et al., 2019, Dingle et al., 2018b, 2020]. If our null expectation is of a large variation in the prior probabilities, then the good correlation can be heuristically justified by a landscape picture [Wales et al., 2003], where $P_B(f|S)$ is interpreted as the “basin volume” $V_B(f)$ (with

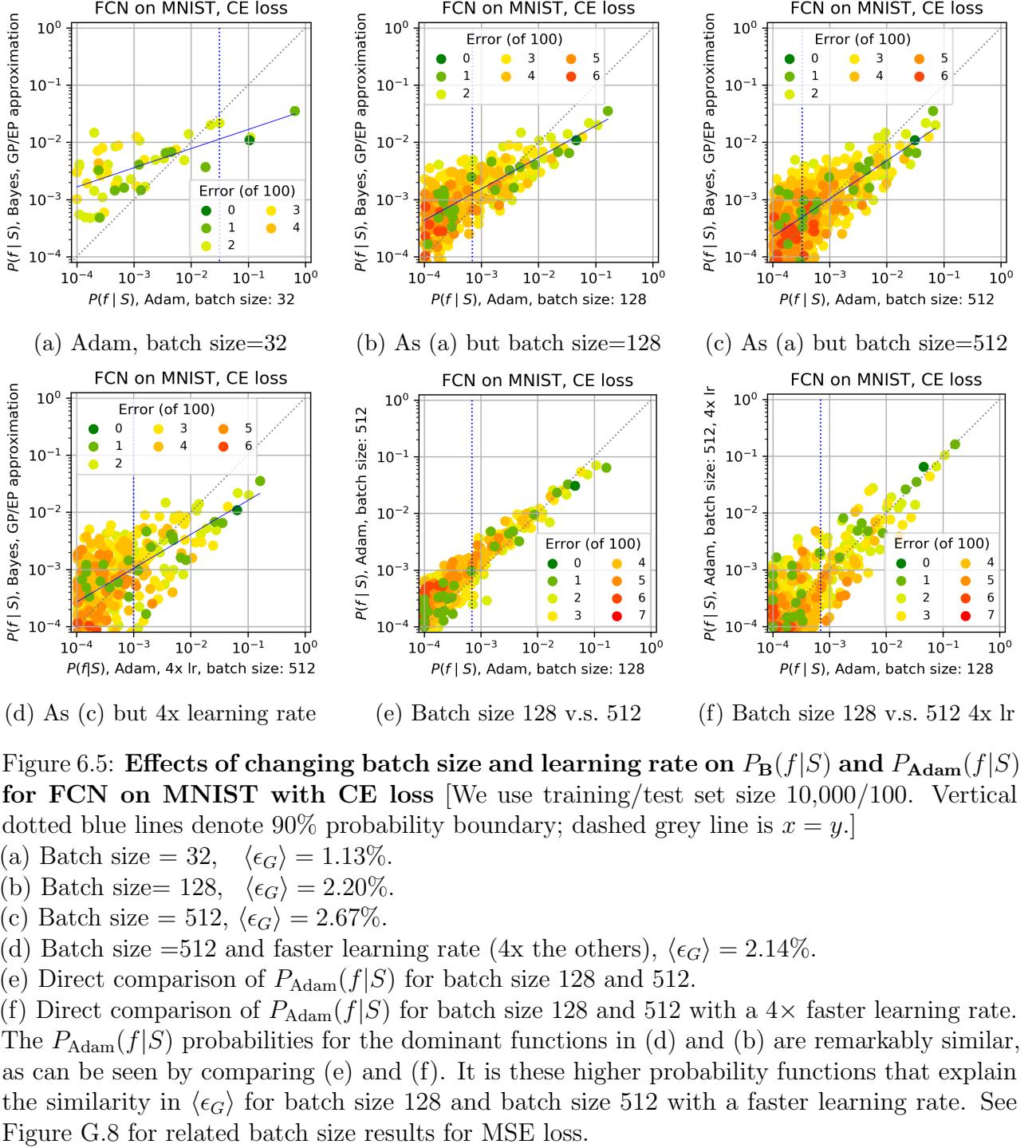


Figure 6.5: **Effects of changing batch size and learning rate on $P_B(f|S)$ and $P_{\text{Adam}}(f|S)$ for FCN on MNIST with CE loss** [We use training/test set size 10,000/100. Vertical dotted blue lines denote 90% probability boundary; dashed grey line is $x = y$.]

- (a) Batch size = 32, $\langle \epsilon_G \rangle = 1.13\%$.
- (b) Batch size= 128, $\langle \epsilon_G \rangle = 2.20\%$.
- (c) Batch size = 512, $\langle \epsilon_G \rangle = 2.67\%$.
- (d) Batch size =512 and faster learning rate (4x the others), $\langle \epsilon_G \rangle = 2.14\%$.
- (e) Direct comparison of $P_{\text{Adam}}(f|S)$ for batch size 128 and 512.
- (f) Direct comparison of $P_{\text{Adam}}(f|S)$ for batch size 128 and 512 with a 4 \times faster learning rate. The $P_{\text{Adam}}(f|S)$ probabilities for the dominant functions in (d) and (b) are remarkably similar, as can be seen by comparing (e) and (f). It is these higher probability functions that explain the similarity in $\langle \epsilon_G \rangle$ for batch size 128 and batch size 512 with a faster learning rate. See Figure G.8 for related batch size results for MSE loss.

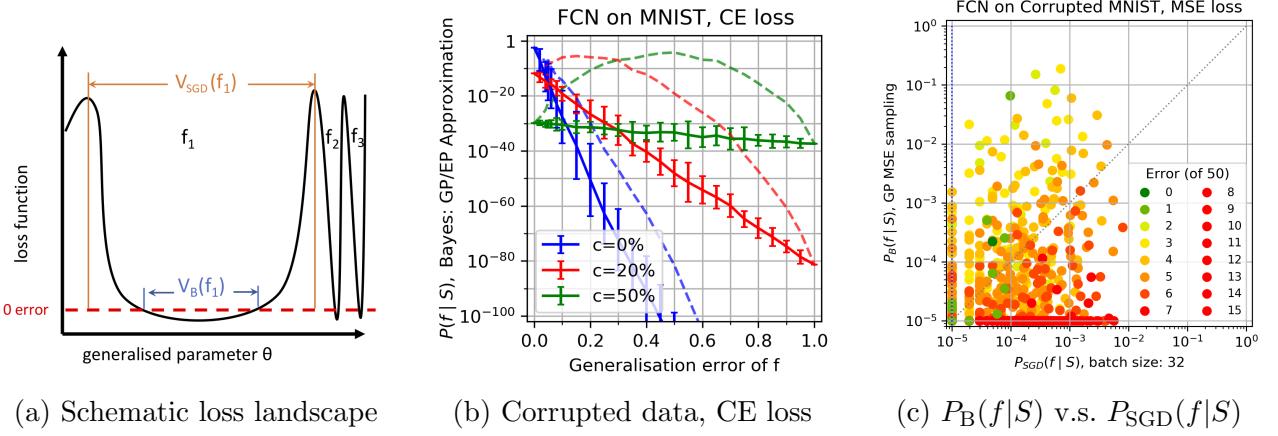


Figure 6.6: Schematic landscape and effects of randomising training labels. (a) Cartoon of a biased loss-landscape. The three functions f_1 , f_2 and f_3 all reach zero classification error (dashed red line), but due to bias in the parameter-function map, the “basin size” $V_B(f_1) \gg V_B(f_2), V_B(f_3)$, which typically implies that for the “basins of attraction” $V_{SGD}(f_1) \gg V_{SGD}(f_2), V_{SGD}(f_3)$. $P_B(f|S)$ is proportional to $V_B(f)$, and $P_{SGD}(f|S)$ is proportional to $V_{SGD}(f)$. (b) $P_B(f|S)$ (solid) and $\rho(\epsilon_G)P_B(f|S)$ (dashed) v.s. ϵ_G , for test set of size 100 and CE loss (as in Figure 6.1b) but including label corruption c . (b) $P_{SGD}(f|S)$ v.s. $P_B(f|S)$ on MNIST with a 2-layer 1024 node wide FCN with MSE loss, test set size 50, and 20% of the training labels randomised ($\langle \epsilon_G \rangle_{SGD} = 13.4\%$ and $\langle \epsilon_G \rangle_{GP} = 5.80\%$). Here functions with frequency < 10 are also shown on the plot. The correlation is much less pronounced than for the unrandomised case shown in Figure 6.1a. Dots on the axes denote functions found by just one of the two methods. Let F be the set of functions found by both the optimiser and under GP sampling. Then $\sum_{f \in F} P_B(f|S) = 99.3\%$, and $\sum_{f \in F} P_{SGD}(f|S) = 24.3\%$. In other words, while the Adam optimiser finds almost all functions with high $P_B(f|S)$, it also finds many functions with low $P_B(f|S)$. The much weaker bias under label corruption observed in (b) likely explains the weaker correlation between the Bayesian results and that of the optimiser found here.

measure $p_{par}(\theta)$) of function f), while $P_{SGD}(f|S)$ is interpreted as the “basin of attraction” $V_{SGD}(f)$, which is loosely defined as a measure of the set of initial parameters θ_i for which the optimiser converges to f with high probability (this concept also found in related form in the dynamical systems literature [Strogatz, 2018]). If $V_B(f)$ varies over many orders of magnitude, then it seems reasonable to expect that $V_{SGD}(f)$ should correlate with $V_B(f)$, as illustrated schematically in Figure 6.6a. Such general intuitions about landscapes are widely held [Wales et al., 2003, Massen and Doye, 2007, Ballard et al., 2017], and have also been put forward for the particular landscapes of deep learning; see in particular Wu et al. [2017] who also argue that functions with good generalisation have larger basins of attraction.

Another source of intuition follows from a well trodden path linking basic concepts from

statistical mechanics to optimisation and learning theory. For example, simple gradient descent (GD) with a small amount of white noise can be described by an over-damped Langevin equation [Welling and Teh, 2011, Smith and Le, 2017, Naveh et al., 2020] that converges (under some light further conditions) to the Boltzmann distribution. The Boltzmann distribution can, in turn, be interpreted as being equivalent to a Bayesian posterior $P_B(f|S) \propto e^{S(f)-\beta E(f)}$ [MacKay and Mac Kay, 2003] where $S(f)$ is configurational “entropy” that counts the number of states that generate f and encodes the prior, and $E(f)$ represents the energy, encoding the log likelihood or loss function. For SGD the equivalent coarse-grained differential equation reduces to Langevin equation with anisotropic noise [Smith and Le, 2017, Zhang et al., 2018] and doesn’t exactly converge to the Bayesian posterior [Mandt et al., 2017, Brosse et al., 2018]. Nevertheless, it has been conjectured that with small step size, SGD may approximate the Bayesian posterior [Naveh et al., 2020, Cohen et al., 2019b], as we empirically find in our experiments. These connections are rich and worth exploring further in this context. Nevertheless, some caution is needed with these analogies to statistical mechanics because they depend on assumptions which may only hold on prohibitively long time-scales.

A better analogy may be to the “arrival of the frequent” phenomenon in evolutionary dynamics [Schaper and Louis, 2014], which, like the “basin of attraction” arguments, does not require steady state. Instead it predicts which structures are likely to be *reached first* by an evolutionary process. For RNA secondary structures, for example, it predicts that a stochastic evolutionary process will reach structures with a probability that to first order is proportional to the likelihood that uniform random sampling of genotypes produces the structure. Indeed, this phenomenon – where the probability upon random sampling predicts the outcomes of a complex search process – can be observed in naturally occurring RNA [Dingle et al., 2015], the result of evolutionary dynamics. This type of non-equilibrium analysis may be more relevant for the way we train most of the DNNs in this chapter, since we stop the first time 0 training error is reached. The analogy between these evolutionary results with what we observe for SGD is intriguing, but needs further exploration.

To illustrate the effect of the amount of bias in the posterior, we randomise labels for MNIST and calculate the $P_B(f|S)$. As we can see in Figure 6.6b, this results in a less strongly biased posterior. The mean log-probability $\langle \log(P_B(f|S)) \rangle$ v.s. ϵ_G curve becomes less steep

with increasing corruption. For a relatively small fraction of low error functions to dominate, as they do for zero corruptions in Figure 6.1a, the bias must be strong enough here to overcome the “entropic” factor $\rho(\epsilon_G)$. For the 20% and 50% corruption this is clearly not the case, and a huge number of functions with larger error will dominate $P_B(f|S)$ and $P_{SGD}(f|S)$. As can be seen in Figure 6.6c, one effect of weaker bias is that the correlation between the optimiser and the Bayesian sampling is much less strong. This behaviour is consistent with the heuristic arguments above, which should only work if the differences in basin volumes are large enough to overcome the myriad other factors that can affect $P_{OPT}(f|S)$.

6.7 Related work on inductive bias on neural networks

In this section we summarise some key aspects of the literature related to why DNNs exhibit good generalisation while overparameterised.

6.7.1 The link between inductive bias and generalisation

Much of the work on inductive biases in stochastic gradient descent (SGD) is framed as a discussion about generalisation. The two concepts are of course intimately related. Before discussing related work on inductive bias DNNs, it may be helpful to distinguish two different questions about generalisation:

- 1) **Question of over-parameterised generalisation:** Why do DNNs generalise at all in the overparameterised regime, where classical learning theory doesn’t guarantee generalisation?
- 2) **Question of fine-tuned generalisation:** Given that vanilla DNNs already generalise reasonably well, how can architecture choice and hyperparameter tuning further improve generalisation?

The first question arises because among the functions that an overparameterised DNN can express, the number that can fit a training data set S , but generalise poorly, is typically many orders of magnitude larger than the number that achieve good generalisation. From classical

learning theory we would therefore expect extremely poor generalisation. However, in practice it is often found that many DNN architectures, as long as they are expressive enough to fit the data, generalise sufficiently well to imply a significant inductive bias towards a small fraction of functions that generalise well.

This question is also related to the conundrum of why DNNs avoid the “curse of dimensionality”, which relates to the poor generalisation that certain highly expressive non-parametric models have in high dimensions [Donoho et al., 2000]. Valle-Pérez et al. [2018] argue that the curse of dimensionality is linked to a prior which is not sufficiently biased and that DNNs may avoid this problem by virtue of the strong bias in the prior.

The second question arises from two common experiences in DNN research. Firstly, changes in architecture can lead to important improvements in generalisation. For example, a CNN with max-pooling typically performs better than a vanilla FCN on image data. Secondly, hyperparameter tuning within a fixed architecture can lead to further improvements of generalisation. While these methods of improving generalisation are important in practice, the starting point is normally a DNN that already has enough inductive bias to raise question 1) above. It is therefore important not to conflate the study of question 2) – as vital as this may be to successful practical implementations — with the more general question of why DNNs generalise in the first place.

6.7.2 Related work on implicit bias in optimiser-trained networks

As mentioned in the introduction, there is an extensive literature on inductive biases in SGD. Much of this literature is empirical: improvements are observed when using particular tuned hyperparameters with variants of SGD. One of the most common rationalisation is in terms of “flatness” which is inspired by early work [Hochreiter and Schmidhuber, 1997a] who predicted that flatter minima would generalise better. Flatness is often measured using some combination of the eigenvalues of the Hessian matrix for a trained DNN. [Keskar et al., 2016] showed that DNNs trained with small batch SGD generalise better than identical models trained with large batch SGD (by up to 5%), and also found a correlation between small batch size and minima that are less “sharp” (using not the eigenvalues of the Hessian but a more computationally tractable sensitivity measure). While these results are genuinely interesting, they are mainly

relevant to issues raised by question 2 above. For example in [Keskar et al., 2016] the authors explicitly point out that their results are not about “overfitting” (e.g. question 1 above).

The effects of changing hyperparameters can be subtle. For example, another series of recent papers [Goyal et al., 2017, Hoffer et al., 2017, Smith et al., 2017] suggest that better generalisation with small batch SGD may be caused by the fact that the number of optimisation steps per epoch decreases when the batch size increases. These studies showed that a similar improvement in generalisation performance to that found by reducing batch size can be created by increasing the learning rate, or by overtraining (i.e. by continuing to train after 100% accuracy has been reached). In particular, in [Hoffer et al., 2017] it was argued that overtraining does not generally negatively impact generalisation, as naive expectations based on overfitting might suggest. These results also challenge some theoretical studies that suggested that SGD may control the capacity of the models by limiting the number of parameter updates [Brutzkus et al., 2017].

In another interesting paper, Zhang et al. [2018] derive a Langevin type equation for both SGD. And argue that in contrast to GD, the noise is anisotropic, and that this may explain why SGD is more likely to find “flatter minima”. Similarly, Jastrzebski et al. [2018] argue that isotropic SGD-induced noise also helps push the optimiser away from sharper minima. An important caveat to the work on sharpness can be found in the work of Dinh *et al.* [Dinh et al., 2017] who use the non-negative homogeneity of the ReLU activation function to show that for a number of the measures used in the papers cited above, the “flatness” can be made arbitrarily large (or sharp) without changing the function (and therefore the generalisation performance) that the DNN expresses. This result suggests that care must be used when interpreting local measures of flatness. Finally in this vein, generalisation has also been linked to related concepts including low frequency [Rahaman et al., 2018], and to sensitivity to changes in the inputs [Arpit et al., 2017, Novak et al., 2018b].

There is much more literature on SGD induced inductive bias, but the upshot is that while fine-tuning optimiser hyperparameters can be very important for improving generalisation, and by implication, the inductive bias of a DNN, a complete understanding remains elusive. Moreover, where improvements are found, these tend to be in the class of answers to question 2) above. An important example of a paper on flatness that does explicitly address question 1

above is [Wu et al., 2017], who show that generalisation trends for data with different levels of corruption correlates with the log of the product of the top 50 eigenvalues of the Hessian both for SGD and for GD trained networks. By heuristically linking their local flatness measure to the global basin volume, they make a very similar argument to the one we flesh out in more detail here, namely that the basin of attraction volume of “good” solutions is much larger than that of “bad” solutions that do not generalise well.

Significant theoretical effort has been spent on extracting properties of a trained neural network that could be used to explain generalisation. By implication, these investigations should also help illuminate the nature of the implicit bias of trained networks. For example, investigators have attempted to use sensitivity to perturbations (whether in inputs or weights) to explain the generalisation performance either using a PAC-Bayesian analysis [Bartlett et al., 2017, Dziugaite and Roy, 2017, Neyshabur et al., 2018a], or a compression approach [Arora et al., 2018, Zhou et al., 2019b]. In contrast to the work described above that studies the specific effect of hyperparameter tuning on SGD, much of the work listed in this paragraph is directly applicable to question 1. A very comprehensive review of this line of work empirically finds that the PAC-Bayesian sensitivity approaches seem the most promising [Jiang et al., 2019], but no clear answer to the question 1 has emerged.

The more theoretical side of the study of SGD has also seen recent progress. For example, [Soudry et al., 2018] showed that SGD finds the max-margin solution in unregularised logistic regression, whilst it was shown in [Brutzkus et al., 2017] that overparameterised DNNs trained with SGD avoid over-fitting on linearly separable data. More recently, [Allen-Zhu et al., 2019] proved agnostic generalisation bounds for SGD-trained DNNs (up to three layers), which impose less restrictive assumptions (on the data, architecture, and optimiser) than previous works. Such theoretical analyses may be a potentially fruitful source of new ideas to explain generalisation.

Another interesting direction is to investigate properties of the loss-landscape itself. Several studies have shown interesting parallels between the loss landscape of DNNs and the energy landscape of spin glasses [Choromanska et al., 2015, Baity-Jesi et al., 2019, Becker et al., 2020]. While such insights may help explain why SGD works so well as an optimiser in these high dimensional spaces, it is at present less clear how these studies help explain question 1) above.

A completely different theme builds on the concept of an information bottleneck [Tishby and Zaslavsky, 2015, Shwartz-Ziv and Tishby, 2017] which suggest that generalisation arises from information compression in deeper layers, aided by SGD. However, recent work [Saxe et al., 2019] suggests that the compression is strongly affected by activation functions used, suggesting again that this approach is not general enough to capture the implicit bias needed to answer question 1. We note that the debate about this theme is ongoing.

Finally, it is important to note that simple vanilla gradient descent (GD), when it can be made to converge, does not differ that much (on the scale of question 1 above) from SGD and its variants in generalisation performance [Keskar et al., 2016, Wu et al., 2017, Zhang et al., 2018, Choi et al., 2019]. Therefore if training with an optimiser itself generates the inductive bias needed to answer question 1, that bias must already largely be present in simple GD.

6.7.3 Related work on implicit bias in random neural networks

We briefly review work inspired by a powerful result from algorithmic information theory (AIT) called the coding theorem [Li and Vitanyi, 2008]. First derived by Levin [Levin, 1974], and building on concepts pioneered by Solomonoff [Solomonoff, 1964], it is closely related to more recent bound applicable to a wider range of input-output maps [Dingle et al., 2018b, 2020]. This bound predicts (under certain fairly general conditions that the maps must fulfil) that upon randomly sampling the parameters of an input-output map M , the probability $P(f)$ of obtaining output f can be bounded as

$$P(f) \leq 2^{-K(f|M)+\mathcal{O}(1)} \approx 2^{-a\tilde{K}(f)+b} \quad (6.4)$$

where $K(f)$ is the Kolmogorov complexity of f , the $\mathcal{O}(1)$ terms do not depend on the outputs (at least asymptotically), $\tilde{K}(f)$ is a suitable approximation to $K(f)$ and a and b are parameters that depend on the map, but not on f . The computable bound was empirically shown to work remarkably well for a wide range of input-output maps from across science and engineering [Dingle et al., 2018b], giving confidence that it should be widely applicable, at least for maps that satisfy the conditions needed for it to apply. In addition, a statistical lower-bound can be derived that predicts that most of the probability weight will lie relatively close to the

bound [Dingle et al., 2020].

The application of this bound to DNNs was first shown in [Valle-Pérez et al., 2018]. We note that the input-output map of interest is not the map from inputs to DNN outputs, but rather the map from the network parameters to the function f it produces on inputs \mathcal{X} which was described in Section 2.1. The prediction of Equation (6.4) for a DNN with parameters sampled randomly (from, for example, truncated i.i.d. Gaussians) is that, if the parameter-function map is sufficiently biased, then the probability of the DNN producing a function f on input data $x_{i=0}^n$ drops exponentially with increasing complexity of the function f . Note that technically we should write f as $f|\mathcal{X}$ to indicate the dependence of the function modelled by the DNN on the inputs \mathcal{X} . We also note that the AIT bound of Equation (6.4) on its own does not force a map to be biased. It still holds for a uniform distribution. But if the map is biased, then it will be biased according to Equation (6.4).

In [Valle-Pérez et al., 2018] it was shown empirically that this very general prediction of Equation (6.4) holds for the $P(f)$ of a number of different DNNs. This testing was achieved both via direct sampling of the parameters of a small DNN on Boolean inputs and with NNGP calculations for more complex systems. In a complementary approach [Mingard et al., 2019] some exact results were proven for simplified networks, that are also consistent with the bound of Equation (6.4). In particular, they proved that for a perceptron with no bias term, upon randomly sampling the parameters (with a distribution satisfying certain weak assumptions), any value of class-imbalance was equally likely. There are many fewer functions with high class imbalance (low “entropy”) than low class imbalance. Low entropy implies low $K(f)$ (but not the other way around). Thus, these results imply a bias of $P(f)$ towards certain simple functions. They also proved that for infinite-width ReLU DNNs, this bias becomes monotonically stronger as the number of layers grows. A different direction was pursued in [De Palma et al., 2018], who showed that, upon randomly sampling the parameters of a ReLU DNN acting on Boolean inputs, the functions obtained had an average sensitivity to inputs which is much lower than if randomly sampling functions. Functions with low input sensitivity are also simple, thus proving another manifestation of simplicity bias present in these systems.

On the other hand, in a recent paper [Yang and Salman, 2019], it was shown that for DNNs

with activation functions such as *Erf* and *Tanh*, the bias starts to disappear as the system enters the “chaotic regime”, which happens for weight variances above a certain threshold, as the depth grows [Poole et al., 2016] (note that ReLU networks don’t have such a chaotic regime). While these hyperparameters are not typically used for DNNs, they do show that there exist regimes where there is no simplicity bias. Note that the AIT coding theorem bound Equation (6.4) still holds, but $P(f)$ is simply approaching a uniform distribution, and the bound becomes loose for small complexity. These results are also interesting because, if the bias becomes weaker, then it may also be the case that the correlation between $P_B(f|S)$ and $P_{SGD}(f|S)$ starts to disappear, an effect we are currently investigating.

6.7.4 Related work comparing optimiser-trained and Bayesian neural networks

Another set of investigations studying random neural networks use important recent extensions of Neal’s seminal proof [Neal, 1994, 2012] – that a single-layer DNN with random i.i.d. weights is equivalent to a Gaussian process (GP) [Mackay, 1998] in the infinite width limit – to multiple layers and architectures [Lee et al., 2017, Matthews et al., 2018, Novak et al., 2018a, Garriga-Alonso et al., 2019, Yang, 2019b]. These studies have used this correspondence to effectively perform a very good approximation to exact Bayesian inference in DNNs. When they have compared them to SGD-trained DNNs [Lee et al., 2017, Matthews et al., 2018, Novak et al., 2018a], the results have generally shown a close agreement between the generalisation performance of optimiser-trained DNNs and their corresponding Bayesian neural network Gaussian process (NNGP).

In this context another significant development is the introduction of the neural tangent kernel (NTK) [Jacot et al., 2018] which approximates the dynamics of an infinite width DNN with parameters that are trained by gradient descent in the limit of an infinitesimal learning rate. Recent comparisons to NNGPs show relatively similar performance of the NTK, see for example [Arora et al., 2019b, Lee et al., 2019, Novak et al., 2019b]. While there are small performance differences, the overall agreement between NNGPs and the NTK or optimiser trained DNNs is close enough to suggest that the primary source of inductive bias needed for

question 1 above is already present in the untrained network, and is essentially maintained under training dynamics.

The linearisation of DNNs offered by NTK can also be used to prove that, in this regime, GD samples from the Bayesian posterior in a sample-then-optimise fashion. For linear regression models, Matthews et al. [2017] showed that solutions after training GD with a Gaussian initialisation correspond to exact posterior samples. This idea is also related to Deep Ensembles which has been proposed to be “approximately Bayesian” in Wilson and Izmailov [2020].

In this context, further indirect evidence comes from Valle-Pérez et al. [2018] who used a simple PAC-Bayesian bound [McAllester, 1999] that applies to exact Bayesian inference, to predict the generalisation error of SGD-trained DNNs. The bound was shown to provide relatively tight predictions for optimiser-trained DNNs for an FCN and CNNs on MNIST, Fashion-MNIST and CIFAR-10. Moreover, this bound, which takes the Bayesian marginal likelihood as input, reproduced trends such as the increase in the generalisation error upon an increased fraction of randomised labels.

These lines of work serve as independent evidence to suggest that optimiser-trained DNNs behave very similarly to the same DNNs trained with Bayesian inference, and helped inspire the work in this chapter, where we directly tackle this question. These studies also suggest that the infinite-width limit may be enough to answer question 1, as the number of parameters in a DNN typically doesn’t have a drastic effect on generalisation (as long as the network is expressive enough to fit the data).

6.7.5 Related work on complexity of data, simplicity bias and generalisation

In Section 6.7.3, we discussed work showing that DNNs may have an inductive bias towards simple functions in their parameter-function map. Here, we briefly discuss how this “simplicity bias” concept may connect to generalisation. As implied by the no free lunch theorem [Wolpert and Waters, 1994], a bias towards simplicity does not automatically imply good generalisation. Instead certain key hypotheses about the data are needed, in particular that it is described by functions that are simple (in a similar sense to the inductive bias). Now the assumption that

a more parsimonious hypothesis is more likely to be true has been influential since antiquity and is often articulated by invoking Occam’s razor. However, the fundamental justification for this heuristic is disputed, see e.g. [Sober, 2015] for an overview of the philosophical literature, e.g. [MacKay, 1992b, Blumer et al., 1987, Rasmussen and Ghahramani, 2001, Domingos, 1999] for a set of different perspectives from the machine learning literature, and e.g. [Rathmanner and Hutter, 2011, Sterkenburg, 2016] for a spirited discussion of the links between the razor and concepts from AIT (pioneered in particular by Solomonoff).

Studies which imply that data typically studied with DNNs is somehow “simple” include an influential paper [Lin et al., 2017] invoking arguments, mainly from statistical mechanics, to argue that deep learning works well because the laws of physics typically select for function classes that are “mathematically simple”, and so easy to learn. More direct studies have also demonstrated certain types of simplicity. For example, following on previous work in this vein, [Spigler et al., 2019] calculated an effective dimension $d_{eff} \approx 15$ for MNIST, which is much lower than the $28^2 = 784$ dimensional manifold in which the data is embedded. Individual numbers can have effective dimensions that are even lower, ranging from 7 to 13 [Hein and Audibert, 2005]. So the functions that fit MNIST data are much simpler than those that fit random data [Goldt et al., 2019]. An implicit bias towards simplicity may therefore improve generalisation for structured data, but it will likely have the opposite effect for more random data.

Chapter 7

Discussion

In this thesis we have explored the question of why DNNs are able to learn functions that generalize beyond the data they used for learning. We have made use of extensive experiments to gain insight into the mechanisms behind generalization, and strengthened the conclusions with theoretical results and predictions.

We can summarize the main results of the thesis as follows

- Result 1. The parameter-function map of most DNN architectures is biased towards functions of low descriptional complexity. For the case of ReLU DNNs in binary classification they are particularly biased towards high class-imbalance functions.
- Result 2. PAC-Bayes theory can be used, in combination with the mean field theory of DNNs, to quantify the bias in the PF map, and to predict learning curves, bound the generalization error, and even predict relative generalization performance between architectures, based on the assumption that the networks are trained using Bayesian inference.
- Result 3. SGD and related stochastic optimizers find solutions with probabilities similar to the Bayesian posterior, justifying the application of our PAC-Bayes theory to models used in practice.

7.1 Simplicity bias in the parameter-function map

Result 1 is covered mainly in Chapter 2 and Chapter 3. In Chapter 2, we thoroughly study a small fully connected feedforward neural network with ReLU activations which can represent Boolean functions of a few (7) inputs. We propose such a system as a useful toy model to study, as the function space is small enough that it can be directly sampled, and one can use efficient measures of complexity for the functions. Using this system, we find that the PF map is exponentially biased towards functions of low complexity, and that this bias is robust to simple hyperparameter changes (like variance of weight distribution and number of layers). This is the same form of simplicity bias predicted from the AIT arguments in Dingle et al. [2018a], when applied to the PF map.

In Section 2.4, we began studying the effects of the bias on learning, by observing different learning metrics, that helped build the intuitions behind the more thorough experiments in Chapter 6. We also look at different complexity measures, and find they give similar qualitative results. The literature on complexity measures is vast, and in Chapter 2 we study only some simple common measures of descriptional complexity. The question of which measures of complexity correlate best with the marginal likelihood (which appears in the PAC-Bayes theorem), or the generalization error, while still being efficiently computable is still an open question. Yang and Salman [2019], inspired by our work, recently showed that the order of the monomials in the Fourier expansion of Boolean functions offer a good complexity measure for Boolean functions, while for continuous functions, we have explored the use the critical sample ratio, proposed in Krueger et al. [2017a]. The work of Yang and Salman [2019] and Bordelon et al. [2020] further highlights that the eigenvalues of the NNGP or NTK kernel of DNNs give us hints about their inductive bias, and thus their intrinsic notions of complexity. The approximations to the marginal likelihood which we use (Section B.3), can also be thought of as complexity measures whose connection with generalization is justified by the PAC-Bayes theorem (Section 4.4), and the experiments in Chapter 5.

In Chapter 3, we study a particular form of simplicity bias: bias towards low entropy Boolean functions (which are those with high class imbalance). In Theorem 3.3.1 in Section 3.3, we prove the existence of an intrinsic bias towards Boolean functions of low entropy in a

perceptron with no threshold bias term, such that $P(T = t) = 2^{-n}$ for $0 \leq t < 2^n$, where T is the number of inputs classified as 1. This result puts an upper bound on the probability that a perceptron with no threshold bias term will be initialised to any particular Boolean function with at least a certain Boolean complexity.

We also study how the entropy bias is affected by the addition of ReLU-activated hidden layers. Theorem 3.4.5 proves that adding layers to a feed-forward neural network with ReLU activations makes the bias towards low entropy stronger. We also show empirically that the bias towards low entropy functions is further increased when a threshold bias term with high enough variance is added. Recently, [Luther and Seung, 2019] have argued that batch normalisation [Ioffe and Szegedy, 2015b] makes ReLU networks less likely to compute the constant function (which has also been experimentally shown in [Page, 2019]). If batch norm increases the probability of high entropy functions, it could help explain why batch norm improves generalisation for (typically class balanced) datasets. We leave further exploration of the effect of batch normalisation on a-priori bias to future work.

We find in Section 3.3.3 that bias within the set of constant $T = t$ functions \mathbb{F}_t is affected by the choice of initialisation, even when the entropy bias is unaffected. This indicates that there are further properties of the parameter-function map that lead to simplicity bias beyond that directly implied by the bias towards low entropy. It also shows that the distribution of parameters can have an effect on $P(f)$, though it is typically small, as we found in Chapter 2. We suggest that the complexity of the conditions on the weights of the perceptron w producing a function should correlate with the complexity of the function, and we conjecture that more complex conditions correlate with a lower probability.

In Section 3.7, we show preliminary results on how bias towards low entropy in $P(f)$ can affect learning in class-imbalanced problems, and comment on the difficulties of analyzing this type of problems.

7.2 Generalization theory of DNNs and marginal-likelihood bound

Result 2 is covered in Chapter 4 and Chapter 5. Chapter 4 presents a unified view on the different approaches towards a generalization theory of deep learning, with a focus on frequentist bounds to the generalization error. We discuss the limitations of worst-case analysis, and the necessity of data dependent and algorithm-dependent analyses. We also present a new PAC-Bayes theorem, which we call the marginal-likelihood PAC-Bayes bound, that bounds the error with high probability over the sampling of the training set, and over the sampling of the Bayesian posterior. We prove that our PAC-Bayes bound has a form that guarantees it is asymptotically optimal (in a sense which we motivate), if the data distribution and priors are such that the learning curve follows a power law.

In Chapter 5, we show results from extensive experiments comparing the true test error of SGD-trained DNNs versus the PAC-Bayes prediction, for a large range of architectures and datasets. We find that the PAC-Bayes bound allows to predict the size of the empirical learning curve exponents, as predicted by our theory (although due to approximations in the marginal likelihood computation, we can only accurately predict relative sizes). These experiments show that our PAC-Bayes theory is a powerful approach to understand the connection between a Bayesian prior in function space $P(f)$, and the generalization performance for some dataset. The results also add more evidence to the thesis that SGD-trained DNNs behave similarly to their Bayesian counterparts, so that a large part of the inductive bias of DNNs can be elucidated from their $P(f)$ at initialization (or with weight sampled according to some uninformative prior). However, there may be exceptions to this, for instance in the case of tanh-like activations, $P(f)$ can be extremely sensitive to the choice of parameter space prior, and our intuitions, and experiments (which have mostly used ReLU networks), may not fully carry through. Studying this class of DNNs is a interesting venue for future research.

The results in Chapter 5 also show that the PAC-Bayes bound, even with the several approximations used, is sensitive enough to predict some of the performance difference between different SOTA architectures in computer vision. Overall, these results place our PAC-Bayes bound as a promising candidate for predicting generalization error without needing a test set.

In particular, we propose that it could be used for neural architecture search by predicting learning curve exponents. However, a more thorough study of this application, comparing it with existing heuristic methods, would be needed to fully understand its possible uses and limitations.

7.2.1 Why the marginal-likelihood PAC-Bayes bound works well

We now discuss several reasons that could explain the success of the marginal-likelihood PAC-Bayes bound which we introduce in Section 4.4, and which suggest ways in which progress in developing new generalization error bounds could be made.

The bound in Section 4.4 is essentially proportional to the Bayesian evidence or marginal likelihood of the model. This directly connects our bound with the Bayesian perspective on generalization. The connection between marginal likelihood and generalization can be traced back to the work of MacKay and Neal [MacKay, 1992a, Neal, 1994, MacKay and Mac Kay, 2003], who empirically found they were correlated and gave heuristic arguments based on Occam’s razor, as well as the early work on PAC-Bayes [Shawe-Taylor and Williamson, 1997, McAllester, 1998] who proved generalization error bounds based on marginal likelihood or similar quantities. Recently, a formal equivalence between marginal likelihood and cross-validation has been shown [Fong and Holmes, 2020], strengthening the connection between marginal likelihood and generalization. Wilson and Izmailov [2020] points out that the marginal likelihood is able to capture the inductive bias of a model, even if the model is arbitrarily flexible. They warn that one should not conflate flexibility (corresponding to the support of the prior, or the “size” of the hypothesis class) with model complexity, and that generalization should take into account the inductive biases as well as the flexibility of the model. Algorithm-independent measures (Section 4.3.1) such as VC dimension are precisely the ones that only capture flexibility, and so miss out on inductive biases. Algorithm-dependent bounds (Section 4.3.2), including our PAC-Bayes bound, are designed to take the inductive bias of the learning algorithm into account, and this is key to their success.

Wilson and Izmailov [2020] as well as the several works in the literature of NNGPs and neural tangent kernels (NTK) [Lee et al., 2017, Jacot et al., 2018, Yang and Salman, 2019] suggest that a function space perspective on neural networks may be fruitful to understand the

behaviour of DNNs, and in particular their generalization, as it is the function encoded by the network which uniquely determines the behaviour of the trained DNN. Our PAC-Bayes bound relies on a prior and posterior in function space, which differs from most PAC-Bayes bounds in the deep learning theory literature which use distributions in parameter space, mostly as a way to measure flatness (see section 4.3.2.1). As we argue in Section 4.4, the function space KL divergence should always provide a bound that is at least as tight as the KL divergence of the corresponding distributions in parameter space, but we may expect that the function space bound may sometimes be much tighter. In particular, if we interpreted our function-space bound in parameter space, we would need to find the KL divergence between the prior and a posterior with support on *the whole region of parameter space with zero training error*. This region is much larger than the individual minima around which the posterior in PAC-Bayes bounds is usually concentrated – for example, exact symmetries when swapping hidden neurons imply combinatorially many “copies” of almost all solutions, and recent results on “mode-connectivity” show that there are many neighbouring solutions which are not accessible by a Gaussian centered around a minimum, because of the topology of the level sets of the loss surface [Garipov et al., 2018]. We expect a distribution supported on a much larger region of parameter space should typically have a much smaller KL divergence with a diffuse prior such as the isotropic Gaussian which we consider, and this could be the main reason why our bound gives much tighter predictions than other PAC-Bayes bounds.

On light of these arguments, we suggest studying bounds (not just PAC-Bayes bounds) which rely on the function-space picture. These could rely on the recent developments on NNGPs and the NTK. Combining function-space with parameter-space analyses (for example to take flatness into account) could also be an interesting direction for future work.

Mingard et al. [2020] have also shown the utility of the function space perspective to study the behaviour of stochastic gradient descent (SGD). They show that SGD, when considered as a sampling process involving initialization followed by stochastic optimization, samples functions with probabilities very close to Bayesian inference. This empirical finding explains why our PAC-Bayes bound is able to predict the learning curve of SGD-trained DNNs, even though the proof only rigorously applies to Bayesian DNNs. More broadly, these findings suggests that studying Bayesian DNNs is a promising direction to understand DNNs as trained in practice.

Our PAC-Bayes bound is also different from other PAC-Bayes bounds in the literature in that it bounds the actual error, with high probability over the Bayesian posterior. This differs from other deterministic PAC-Bayes bounds in the literature which bound the average error uniformly over a large class of posteriors [Nagarajan and Kolter, 2019]. This new result suggests that the quantifier reversal lemma of McAllester [1998] can give qualitatively different kinds of high-probability bounds than those obtained from the more commonly used PAC-Bayesian model-averaging bounds developed later McAllester [1999], and we believe that it could offer interesting insights into obtaining high-probability bounds not based on uniform (or non-uniform) convergence.

Nagarajan and Kolter [2019] also showed that a large class of doubled-sided bounds are fundamentally unable to predict the generalization for SGD-trained DNNs, at least for certain synthetic datasets. On the other hand, our bound assumes realizability which automatically allows for an analysis using one-sided bounds, rather than double-sided bounds – i.e. bounds in the generalization error rather than bounds on the absolute value of the generalization gap (see e.g. the realizable PAC bound for a simple example [Shalev-Shwartz and Ben-David, 2014]). This fact, and the good results of our bound, suggest that realizability may be an important property of successful bounds for deep learning.

7.3 SGD-trained networks are approximately Bayesian

Result 3 is studied in Chapter 6. In that chapter we show evidence that, on a log scale, $P_{\text{OPT}}(f|S) \approx P_B(f|S)$, for a large variety of optimizers, architectures, and training datasets. By studying $P_{\text{OPT}}(f|S)$ for different algorithms, we find that there are also measurable second order deviations that are sensitive to hyperparameter tuning and optimiser choice. While they may not explain the fundamental question of why DNNs generalize, second order deviations from $P_B(f|S)$ are important in practice for further fine-tuning the generalisation performance.

For the conundrum of why DNNs generalise at all in the overparameterised regime, these results strongly suggest that the solution must be found in the properties of $P(f)$, and not in biases introduced by SGD. On the other hand, the results in the other chapters show that DNN priors are exponentially biased towards simple functions, which can help explain the

inductive bias of $P(f)$. However, understanding the properties of $P(f)$ in more detail for different architectures, and its relation to generalization (as given for example by the PAC-Bayes theorem), for different datasets, is still an area where much work is needed.

Our function probability perspective also provides more fine-grained tools for the analysis of DNNs than simply comparing the average test error. This picture could facilitate the investigation of hyperparameter changes, or potentially also the study of techniques such as batch normalisation or dropout. It could assist in the design of new architectures or optimisers.

It is not obvious how to determine the epistemic uncertainty in a prediction of a DNN model. However, if, as we argue in Chapter 6, SGD behaves like a Bayesian sampler, then this offers a strong justification for using methods like Deep Ensembles to measure Bayesian uncertainty in the case of DNNs [Wilson and Izmailov, 2020].

Most of the examples we have studied are for image classification. It would be interesting to study the related problem of using DNNs for regression, as well as more examples on sequence data. Sampling considerations means that it is easier to study $P_{\text{SGD}}(f|S)$ for smaller generalisation errors. It would be interesting to study systems with intrinsically larger $\langle \epsilon_G \rangle$ within this picture as well. There the biasing effect of the optimiser may be larger.

Finally, to study the correlation between $P_B(f|S)$ and $P_{\text{SGD}}(f|S)$ in Chapter 6, we mainly used a fixed test and training set. While we did examine other test and training sets (see Chapter G), this was mainly to confirm that our results were not an artefact of our particular choices. A promising future direction would be a Bayesian approach that includes averaging over training sets.

7.4 Conclusion

In conclusion, the main three results in this thesis offer substantive evidence that the mechanism behind generalization in DNNs is the simplicity bias in their parameter-function map. The origin of the simplicity bias still holds many questions. We have shown that it can be connected to arguments from AIT and to a form of low-entropy bias for the case of the perceptron. However, even for simple DNNs, like those we study in Chapter 2, we still don't fully understand why the bias is so strong (and why it displays a Zipfian behaviour). Our experiments show that

the MFT of DNNs offers a good approximation of the function-space prior $P(f)$ for realistic DNNs, and that it is a good starting point to understand the simplicity bias more precisely, and quantitatively, even for more complex architectures.

To understand the relationship between the bias encoded in $P(f)$ and generalization, we have developed a PAC-Bayes theory based on marginal likelihood of DNNs, and shown using theory and extensive experiments that it is a good candidate for a predictive theory of generalization and learning curves. The PAC-Bayes theorem captures in a precise form the intuition that if the inductive bias agrees with the true data distribution, we are likely to generalize. It doesn't answer, however, the deeper question of why problems in practice agree with the bias of DNNs. At a high level, there have been arguments that propose that real-world problems may also be biased towards simplicity [Schmidhuber, 1997, Lin et al., 2017], or even that they are biased towards similar types of simplicity that neural networks are [Bengio et al., 2007], but a better understanding of the distribution of real-world problems (or of the extent to which it may even be possible to characterize this distribution) is still lacking.

Finally, we have shown that common optimizers of DNNs approximate Bayesian inference, justifying the use of our generalization analysis for real-world optimizer-trained DNNs, as well as other Bayesian deep learning methods used in practice, like deep ensembles.

We think that these results help elucidate the fundamental reasons that make deep learning systems generalize in the overparameterized regime, one of the main open questions in the current theory of machine learning, and hope that our results inspire research into the many further questions we ask in the discussion above, as well as help in the practical development of better deep learning systems.

Appendices

Appendix A

Experiment details

A.1 Experiment details for Chapter 2

In the main experiments of the paper we used two classes of architectures. Here we describe them in more detail.

- Fully connected networks (FCs), with varying number of layers. The size of the hidden layers was the same as the input dimension, and the nonlinearity was ReLU. The last layer was a single Softmax neuron. We used default Keras settings for initialization (Glorot uniform).
- Convolutional neural networks (CNNs), with varying number of layers. The number of filters was 200, and the nonlinearity was ReLU. The last layer was a fully connected single Softmax neuron. The filter sizes alternated between (2, 2) and (5, 5), and the padding between SAME and VALID, the strides were 1 (same default settings as in the code for Garriga-Alonso et al. [2018]). We used default Keras settings for initialization (Glorot uniform).

In the experiments where we learn Boolean functions with the smaller neural network with 7 Boolean inputs and one Boolean output (results in Figure 2.3a), we use a variation of SGD similar to the method of adversarial training proposed by Ian Goodfellow Goodfellow et al. [2014]. We chose this second method because SGD often did not find a solution with 0

training error for all the Boolean functions, even with many thousand iterations. By contrast, the adversarial method succeeded in almost all cases, at least for the relatively small neural networks which we focus on here.

We call this method *adversarial SGD*, or *advSGD*, for short. In SGD, the network is trained using the average loss of a random sample of the training set, called a *mini-batch*. In advSGD, after every training step, the classification error for each of the training examples in the mini-batch is computed, and a moving average of each of these classification errors is updated. This moving average gives a score for each training example, measuring how “bad” the network has recently been at predicting this example. Before getting the next mini-batch, the scores of all the examples are passed through a softmax to determine the probability that each example is put in the mini-batch. This way, we force the network to focus on the examples it does worst on. For advSGD, we also used a batch size of 10.

In all experiments we trained with a learning rate of 0.01, and early stopping when the accuracy on the whole training set reaches 100%, we used binary cross entropy as the loss function. We found that Adam could learn Boolean functions with the smaller neural network as well, but only when choosing the mean-squared error loss function.

A.1.1 Finite-size effects for sampling probability

Since for a sample of size N the minimum estimated probability is $1/N$, many of the low-probability samples that arise just once may in fact have a much lower probability than suggested. See Figure A.1), for an illustration of how this finite-size sampling effect manifests with changing sample size N . For this reason, these points are typically removed from plots.

A.2 Experimental details for Chapter 5

Here we describe in more detail the experiments carried out in Chapter 5. The results we plot for both test errors and PAC-Bayes bounds are averaged over 8 random initializations and samples of training set, except for some of the large m points for which we didn’t do this due to limited compute time. We find that the variance in both the error and PAC-Bayes bound is very small and decreases for large m , as is expected from concentration inequalities (at least

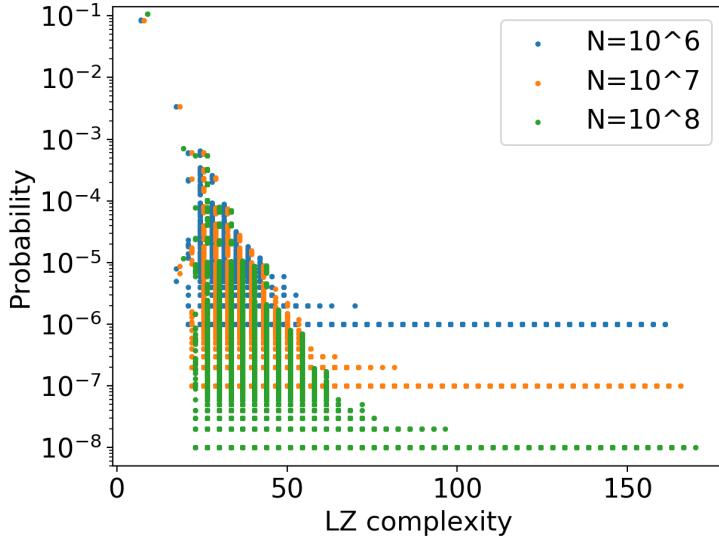


Figure A.1: Probability (calculated from frequency) versus Lempel-Ziv complexity for a neural network of shape $(7, 40, 40, 1)$, and sample sizes $N = 10^6, 10^7, 10^8$. The lowest frequency functions for a given sample size can be seen to suffer from finite-size effects, causing them to have a higher frequency than their true probability.

for the test error).

A.2.1 Training, dataset, and architecture details

Training For all the experiments, we train the networks with Adam, cross-entropy loss, using either batch size 32 or 256, with a learning rate of 0.01, and we train until the first time we reach exactly 0 training error. The exception is VGG and CNN networks for which we used a learning rate between 1e-5 and 1e-5, which was necessary in order to converge to 0 training error.

Datasets. We use binarized versions of five datasets: MNIST [LeCun and Cortes, 2010], Fashion-MNIST [Xiao et al., 2017], EMNIST (ByClass split) [Cohen et al., 2017], KMNIST [Clanuwat et al., 2018], and CIFAR10 (which we refer to as CIFAR) [Krizhevsky et al.]. The datasets were binarized by assigning label 0 to the first half of classes, and label 1 to the second half. The input images are normalized to lie in the range $[0, 1]$.

Architectures We studied the following architectures

- Fully connected networks (FCs), with varying number of layers. The size of the hidden

layers was 1024, and the nonlinearity was ReLU. The last layer was a single Softmax neuron. If the number of layers is not specified, we use 2 hidden layers.

- Convolutional neural networks (CNNs), with varying number of layers. The number of filters was 1024, and the nonlinearity was ReLU. The output layer was a single Softmax neuron fully connected to a flattening layer. The filter sizes alternated between (2, 2) and (5, 5), and the padding between SAME and VALID, the strides were 1 (same default settings as in the code for Garriga-Alonso et al. [2018]). We used CNNs without pooling, and with two kinds of pooling: avg and max. Pooling was applied at every layer with a pooling size of 2 and padding SAME, and globally at the last layer, before flattening. If the number of hidden layers is not specified, we use 4 hidden layers.
- vgg16. Keras implementation (<https://keras.io/api/applications/vgg/#vgg16-function>)
Simonyan and Zisserman [2015]
- vgg19. Keras implementation (<https://keras.io/api/applications/vgg/#vgg19-function>)
Simonyan and Zisserman [2015]
- resnet50. Keras implementation (<https://keras.io/api/applications/resnet/#resnet50-function>)
He et al. [2016a]
- resnet101. Keras implementation (<https://keras.io/api/applications/resnet/#resnet101-function>)
He et al. [2016a]
- resnet152. Keras implementation (<https://keras.io/api/applications/resnet/#resnet152-function>)
He et al. [2016a]
- resnetv2_50. Keras implementation (<https://keras.io/api/applications/resnet/#resnet50v2-function>). He et al. [2016b]
- resnetv2_101. Keras implementation (<https://keras.io/api/applications/resnet/#resnet101v2-function>). He et al. [2016b]
- resnetv2_152. Keras implementation (<https://keras.io/api/applications/resnet/#resnet152v2-function>). He et al. [2016b]

- resnext50. Keras applications implementation (https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnext.py). Xie et al. [2017]
- resnetxt101. Keras applications implementation (https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnext.py). Xie et al. [2017]
- densenet121 Keras implementation (<https://keras.io/api/applications/densenet/#densenet121-function>). Huang et al. [2017]
- densenet169 Keras implementation (<https://keras.io/api/applications/densenet/#densenet169-function>). Huang et al. [2017]
- densenet201 Keras implementation (<https://keras.io/api/applications/densenet/#densenet201-function>). Huang et al. [2017]
- mobilenetv2. Keras implementation (<https://keras.io/api/applications/mobilenet/#mobilenetv2-function>). Howard et al. [2017]
- nasnet. Keras implementation (<https://keras.io/api/applications/nasnet/#nasnetlarge>). Zoph et al. [2018]

For vgg16,vgg19,resnets ,densenets, mobilenetv2, and nasnet, we used average global pooling in the last layer, unless otherwise specified. All networks weights, except normalization layer parameters, were initialized from a Gaussian with variance $\sigma_w^2 = 1.41$, and the biases were initialized with variance $\sigma_b^2 = 0$. Normalization layers were initialized using their Keras default initialization.

A.3 Further detail for experiments in Chapter 6

In this Appendix we provide further details and explanation of the methodology outlined in Section 6.3. In particular, we describe the datasets in Section A.3.1, the architectures in Section A.3.2, and discuss our experiments in Section A.3.3. For the details on the GP approximation for $P_B(f|S)$ see Chapter B.

A.3.1 Data sets

To efficiently sample functions, we use relatively small test sets (typically $|E| = 100$) and, as is often done in the theoretical literature, binarise our classification datasets. We define the datasets used below:

MNIST: The MNIST database of handwritten numbers [LeCun et al., 1999] was binarised with even numbers classified as 0 and odd numbers as 1. Unless otherwise specified, we used $|S| = 10000$ and $|E| = 100$.

Fashion-MNIST: The Fashion-MNIST database [Xiao et al., 2017] was binarised with T-shirts, coats, pullovers, shirts and bags classified as 0 and trousers, dresses, sandals, trainers and ankle boots classified as 1. Unless otherwise specified, we used $|S| = 10000$ and $|E| = 100$.

IMDb movie review dataset: We take the IMDb movie review dataset from Keras. The task is to correctly classify each review as positive or negative given the text of the review. We preprocess the set by removing the most common words and normalising.¹ This procedure was employed to make sure there are functions with high enough probability to be sampled multiple times with the experiments in Section A.3.3.1 and Section A.3.3.2. Used with $|S| = 45000$ and $|E| = 50$.

Ionosphere Dataset: This is a small non-image dataset with 34 features² aimed at identifying structure in the ionosphere [Sigillito et al., 1989]. Used with $|S| = 301$ and $|E| = 50$.

For image datasets, we will typically use normalised data (pixel values in range [0,1]) for MSE loss, and unnormalised data for CE loss (pixel values in range [0,255]).

A.3.2 Architectures

We used the following standard architectures.

FCN: 2 hidden layer, 1024 node vanilla fully connected network (FCN) with ReLU activations.

CNN + (Max Pooling) + [BatchNorm]: Layer 1: Convolutional Layer with 32 features size 3×3 . (Layer 1a: Max Pool 2×2). [Layer 1b: Batch Norm]. Layer 2: Flatten. Layer 3: FCN with width 1024. [Layer 3a: Batch Norm]. Layer 4: FCN, 1 output with ReLU

¹We used the version of the dataset and preprocessing technique given here: <https://www.kaggle.com/drscarlat/imdb-sentiment-analysis-keras-and-tensorflow>

²<https://archive.ics.uci.edu/ml/datasets/Ionosphere>

activations.

LSTM: Layer 1: Embedding layer. Layer 2: LSTM, 256 outputs. Layer 3: FCN, 512 outputs. Layer 4: Fully-Connected, 1 output with ReLU activations for the fully connected layers. Hyperparameters are, unless otherwise specified, the default values in Keras 2.3.0. See Section A.3.3.1 for details on the parameter initialisation.

A.3.3 Methodology in detail

For each experiment performed in Chapter 6, we pick a DNN \mathcal{N} (either FCN, CNN, or an LSTM) and a dataset \mathcal{D} (MNIST, Fashion-MNIST or the IMDb dataset). We also pick a fixed training set $S \subset \mathcal{D}$, and a fixed test set $E \subset \mathcal{D}$. Training sets are typically of size 10,000 for FCN and CNN and 45,000 for the LSTM. Test sets are typically small, 100 for the FCN and CNN, and 50 for the LSTM.

A.3.3.1 Using an optimiser to calculate $P_{\text{OPT}}(f|S)$

When calculating $P_{\text{OPT}}(f|S)$ we first pick an optimiser OPT which is either plain SGD, or one of its derivatives: Adam, Adagrad, RMSprop, or Adadelta. Next we pick a loss-function, either mean-square error (MSE) or cross-entropy (CE). We also need to pick an initial parameter distribution $\tilde{P}_{\text{par}}(\theta)$ which we take to be from a truncated i.i.d. Gaussian distribution (the distribution from which the DNN \mathcal{N} is randomly initialised, see Equation (6.3)).

Algorithm 1 Calculating $P_{\text{OPT}}(f|S)$

```

input: DNN  $\mathcal{N}$ , training data  $S$ , test data  $E$ , optimiser  $OPT$ .
 $F \leftarrow \langle \rangle$  {the ‘functions’ found during training}
do  $n$  times:
    re-initialise the weights of  $\mathcal{N}$  from an i.i.d. Gaussian distribution
    train  $\mathcal{N}$  on  $S$  until it reaches 100 % training accuracy
    record the classification of  $\mathcal{N}$  on  $E$  and save it to  $F$ 
     $A \leftarrow \emptyset$  {the frequency and ‘volume’ of each ‘function’}
    for each distinct  $f \in F$  do
        let  $\rho_f$  be the frequency of  $f$  in  $F$ 
        calculate the probability  $P_{\text{OPT}}(f|S) = \rho_f/n$  of  $f$  in  $F$ 
        save  $P_{\text{OPT}}(f|S)$  to  $A$ 
    end for
    return  $A$ 

```

For fully-connected layers we used $\sigma_b = 0$ and $\sigma_w = 1/\sqrt{w}$ where w is the width of the layer. For convolutional and LSTM layers we used the default initialisation provided by Keras 2.3.0³. This specifies $\tilde{P}_{par}(\theta)$. As we see in Algorithm 1, we then sample n times from $\tilde{P}_{par}(\theta)$, training each time with OPT until the training error on S is zero, at which point we record the function by what errors it makes on E . Note that if for some reason SGD does not converge, we don't count that run in order to have normalised distributions over functions. We typically run between $n = 10^4$ and $n = 10^7$ runs (depending on the system). Once the runs are finished, we compile all the empirical frequencies for the functions that are found.

A.3.3.2 Bayesian sampling for $P_B(f|S)$

We use the GP approximation to estimate the Bayesian posterior $P_B(f|S)$. Here we follow [Valle-Pérez et al., 2018] where this technique is explained (see also Chapter B for more details). We need to define a distribution $P_{par}(\theta)$ (the definition of the prior from which we calculate $P_B(f|S)$, see Equation (6.1)). While there are some subtleties in how the prior distribution $P_{par}(\theta)$ relates to the initialisation distribution $\tilde{P}_{par}(\theta)$, we took a simple approach and defined P_{par} to be the same as the corresponding $\tilde{P}_{par}(\theta)$, except we set σ_b to be a small constant, typically $0.1 \times \sigma_w$.

To estimate $P_B(f|S)$, there is a small compromise that must be made here. For a number of reasons, MSE loss is less popular for the kinds of classification problems we mainly study in this paper. We also find that it typically takes significantly longer to train using an optimiser so that $P_{OPT}(f|S)$ is more expensive to evaluate for MSE loss on the problems we study. On the other hand, $P_B(f|S)$ can be directly sampled n times from the exact posterior (described in Section B.3.2) using Algorithm 2), and so is relatively accurate and simple to evaluate.

For CE loss, which is more frequently used for classification, and is also typically quicker to train than MSE for SGD and its variants, we need to use a further approximation. Here we follow [Valle-Pérez et al., 2018] and use the expectation-propagation (EP) approximation for $P_B(f|S)$. We then estimate the posterior log probabilities using the estimations of the log marginal likelihoods $\log P(S)$ (see Section B.3.1 for explanation) in Algorithm 3, or we sample

³CNN: https://keras.io/api/layers/convolution_layers/convolution2d/
LSTM: https://keras.io/api/layers/recurrent_layers/lstm/

from the approximate EP posterior and use Algorithm 2. These two methods give very similar answers (Figure B.1c).

Algorithm 2 Calculating $P_B(f|S)$ (via sampling)

```

input: DNN  $\mathcal{N}$ , training data  $S$ , test data  $E$ .
 $F \leftarrow \emptyset$  {functions sampled from the GP or GP/EP posterior}
do  $n$  times:
    sample a function  $f$  from the GP or GP/EP posterior when conditioning on  $S$ 
    find  $f$  on  $E$ 
    save  $f$  to  $F$ 
 $\mathfrak{R} \leftarrow \emptyset$  {function probabilities}
for each distinct  $f \in F$  do
    let  $\rho_f$  be the frequency of  $f$  in  $F$ 
    calculate  $P_B(f|S) = \rho_f/n$ 
    save  $P_B(f|S)$  to  $\mathfrak{R}$ 
end for
return  $\mathfrak{R}$ 

```

Algorithm 3 Calculating $P_B(f|S)$ for specific f via the ratio of likelihoods approximation

```

input: DNN  $\mathcal{N}$ , training data  $S$ , test data  $E$ , optimiser  $OPT$ , set of functions  $F$  (from
Algorithm 1)
for each distinct  $f \in F$  do
    let  $\rho_f$  be the frequency of  $f$  in  $F$ 
    use GP or GP/EP approximation to estimate  $P_B(f|S)$  of  $f$  using the ratio of likelihoods
    approximation.
    save  $P_B(f|S)$  to  $A$ 
end for
return  $A$ 

```

A.3.3.3 Calculating $P_B(f|S)$ for functions with a wider range of ϵ_G

Given a training dataset S and test dataset E we can generate a random sample of different partial functions with varying levels of error on E (by taking the test set classification and corrupting some percentage of labels). We can then use the GP/EP approximation to estimate $P_B(f|S)$. We typically sample 20 examples for each number of errors. The averages are taken on the logs of the probabilities, and error bars on plots are 2σ , where σ is the standard deviation. Note that the vast majority of functions have such small probabilities, that it is not feasible to estimate their $P_{OPT}(f|S)$ (nor use Algorithm 2)

While the experiments will be informative for how the space is biased, it does not guarantee that all high-probability functions will be found. Since these functions affect generalisation the most, we rely on the results from Algorithm 2 to check that there are no high-probability functions that are missed.

Algorithm 4 Calculating $P_B(f|S)$ for larger range of ϵ_G

```

input: DNN  $\mathcal{N}$ , training data  $S$ , test data  $E$ .
for  $\epsilon \in \{0.0, 0.5, \dots 1.0\}$  do
     $V_\epsilon \leftarrow \langle \rangle$ 
    generate classification  $c$  with error  $\epsilon$  on  $E$  (by randomly choosing  $|E| \times \epsilon$  distinct labels in
    the correct function (restricted to the test set) to switch to incorrect).
    use GP/EP and “ratio of likelihoods” approximation to estimate the  $P_B(f|S)$  of  $c$ 
    save the relative volume  $P_B(f|S)$  of  $f$  to  $V_\epsilon$ 
end for
return  $V_{0.0} \dots V_{1.0}$ 

```

We here make a few more remarks for interpreting the results in experiments where we generate functions for a wide range of errors, such as those in Figure 6.1b. Firstly, as shown in Figure G.16a in Section G.4, there can be a wide variation in the probabilities p_i of misclassifying each of the 100 images in this test set. The generalisation error is therefore dominated by a small number of harder to classify images. This means that the probabilities of functions for a fixed ϵ_G can vary a lot. This explains why we find that even though the highest probability function in Figure 6.1a is a 1-error function, on average the probability for 1-error functions in Figure 6.1b is lower than that of the 0-error function. The high variance in $P_B(f|S)$ within the functions of fixed ϵ_G , as well as the EP approximation also means that the estimates of $\langle P_B(f|S) \rangle_{\epsilon_G}$ may be less accurate. For Figure 6.1b, $\sum_\epsilon \rho(\epsilon) \langle P_B(f|S) \rangle_{\epsilon_G} \approx 0.1$, which is not far off the correct value of 1. Keeping in mind that we may be missing some higher probability outliers in the average due to finite sampling, this agreement is encouraging. In short, although the quantitative values may not be fully accurate, the results in Figure 6.1b are indicative of the strong exponential trend towards low probability with increasing error.

A third point of clarification is that in Figure 6.1b we used CE loss rather than MSE loss. We made this choice because we need to estimate very small $P_B(f|S)$ values, for which we need to use the “ratio of likelihoods” approximation. While we only described how to use EP for CE loss, we could also use the EP approximation to estimate $P_B(f|S)$ for the analytical posterior

of the GP with MSE loss. However, from some preliminary tests, the EP approximation introduced significant systematic errors and in particular it didn't show better results than the EP/“ratio of likelihoods” method for CE loss. In Section B.4, we can also see that the differences between $P_B(f|S)$ for MSE and CE loss are probably of less than a few orders of magnitude, and would not significantly affect the results in Figure 6.1b.

A.3.3.4 Further notes on methods

When $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$ are obtained by sampling, we typically sample between 10^5 and 10^7 times. To avoid finite sampling effects, we place any functions found with frequencies < 10 on the axes of our graphs. However, those functions are included in calculations involving generalisation errors, although typically they contribute very little because they are by definition low probability.

We will also regularly provide values for $\sum_{f \in F} P_B(f|S)$ and $\sum_{f \in F} P_{\text{OPT}}(f|S)$ in our figures comparing $P_{\text{OPT}}(f|S)$ with $P_B(f|S)$ (only when $P_B(f|S)$ is obtained by direct sampling), where F is the set of functions found by both GP sampling and by the optimiser (within a finite number of samples from the GP and optimiser-trained DNNs, typically $10^4 - 10^6$). A value of $\sum P_B(f|S)$ close to 1 indicates that almost all functions with high $P_B(f|S)$ are found by the optimiser. Similarly, a high value of $P_{\text{OPT}}(f|S)$ implies that GP sampling finds almost all functions with high $P_{\text{OPT}}(f|S)$ found by the optimiser. Note that for the EP approximation needed for CE loss, we directly calculate the $P_B(f|S)$ for functions found by the optimiser, and so a $\sum P_{\text{OPT}}(f|S)$ is not defined.

Finally, when values for the generalisation error $\langle \epsilon_G \rangle$ are depicted in the graphs showing $P_B(f|S)$ v.s. $P_{\text{OPT}}(f|S)$, $\langle \epsilon_G \rangle$ refers to the generalisation error of the optimiser. When $\langle \epsilon_G \rangle$ is presented in graphs with only $P_B(f|S)$ it refers to the $\langle \epsilon_G \rangle$ from GP sampling, or alternatively we use the symbol $\langle \epsilon_G \rangle_{GP}$.

Appendix B

Mean field theory of DNNs

In this appendix we will briefly review the mean field theory of DNNs. This theory establishes a connection between Bayesian DNNs with infinitely many hidden neurons per layers and Gaussian processes known as neural network Gaussain processes (NNGPs) [Lee et al., 2018]. It is called a mean field theory, because in this limit the prior distributions of the hidden neurons become uncorrelated [Schoenholz et al., 2017, Lee et al., 2018, Yang and Schoenholz, 2018].

B.1 Bayesian framework

The theory of Gaussian processes (including NNGPs) is based on the Bayesian formulation to learning, which we will review in this section.

Bayesian learning begins with a hypothesis space, which in our case we will take to be the space of functions that a DNN can implement $\mathcal{F} \subseteq \mathcal{Y}^{\mathcal{X}}$, following the notation in Section 2.1. A probabilistic distribution P called the *prior* is defined in that space, representing our *a priori* belief that different hypotheses are true.

The next ingredient is an observation space, which for supervised learning corresponds to the set of training sets $\mathcal{S} = (\mathcal{X} \times \mathcal{Y})^m$ corresponding to m observed pairs of inputs and outputs. A *likelihood function* $P(S|f)$, represents the conditional probability of different training sets $S \in \mathcal{S}$ given a function $f \in \mathcal{F}$ is true. This can be decomposed as $P(S|f) = P(\mathbf{Y}|f, \mathbf{X})P(\mathbf{X}|f)$, where \mathbf{Y} and \mathbf{X} are the vector of outputs and inputs in the training set S . In this thesis we assume, as it is often done for Gaussian processes, that $P(\mathbf{X}|f) = P(\mathbf{X})$ is independent of f ,

and thus we will see that it does not affect the posterior. Because of this we will abuse notation and refer to $P(\mathbf{Y}|f, \mathbf{X})$ as $P(S|f)$ where the dependence on \mathbf{X} is left implicit for brevity.

In the idealized case, the prior truly represents our best guess at the probabilities of different hypotheses. In that case, after observing some data S , we can use basic rules of conditional probabilities to obtain Bayes' theorem, which for the supervised learning likelihood becomes

$$P(f|S) = \frac{P(\mathcal{Y}|f, \mathbf{X})P(\mathbf{X})P(f)}{\sum_f P(\mathcal{Y}|f, \mathbf{X})P(\mathbf{X})} \quad (\text{B.1})$$

Using the abuse of notation $P(S|f) := P(\mathcal{Y}|f, \mathbf{X})$ and $P(S) := \sum_f P(\mathcal{Y}|f, \mathbf{X})$ we have the simpler expression

$$P(f|S) = \frac{P(S|f)P(f)}{P(S)} \quad (\text{B.2})$$

$P(f|S)$ is called the *posterior* and represents the probability of different hypotheses based on our prior belief $P(f)$ and the observed data S . The denominator, $P(S)$, is called the *marginal likelihood*, which is crucial in the computation of the PAC-Bayesian bounds of Section 4.4 used in Chapter 5.

The prior for DNNs. In the case of Bayesian DNNs the prior over functions $P(f)$ is induced by a prior over parameters $P_{\text{par}}(\theta)$ as introduced in Section 2.3 and in Section 6.2.1. Gaussian processes instead define a prior over functions directly, by assuming it takes a specific form, as we will see in the next section.

The likelihood for classification. In this thesis we study classification without label noise, so that the labels in the training set correspond to the true labels which we wish to predict. In this case, \mathcal{Y} is discrete, and we use a 0 – 1 or classification error likelihood. Formally, if $S = \{(x_i, y_i)\}_{i=1}^m$ corresponds to the set of training pairs, then we let

$$P(S|f) = \begin{cases} 1 & \text{if } \forall i, f(x_i) = y_i \\ 0 & \text{otherwise .} \end{cases}$$

With these ingredients, Bayesian learning amounts to computing the posterior distribution for a given prior, and observed data. For Gaussian processes with Gaussian likelihood, this can be done analytically. Because of this, we will use mean-squared error likelihood for classification

in some experiments in Chapter 6. This is described in Section B.3.2. For non-Gaussian likelihoods, like the $0 - 1$ likelihood, the marginal likelihood is typically not analytically tractable, so we use approximations, like the expectation-propagation (EP) approximation, as we describe in Section B.3.1.

B.2 Neural network Gaussian processes

The mean field theory of DNNs concerns itself with the prior over the pre-activation outputs $\tilde{P}(h^{(L+1)})$ expressed by DNNs, where $h^{(L+1)}$ is a real valued function of x given by the pre-activation of the last layer (Theorem 2.1.1), which is itself a random variable, when the parameters are random, which is the case for Bayesian DNNs. The main assumption in the theory is that the distribution over parameters P_{par} is such that the weights and biases are both distributed i.i.d Gaussian and are also independent of each other (but may have different variances), though slightly more relaxed assumptions in the parameter distributions are possible. Note that this is slightly different from $P(f)$ when we consider a DNN with binary outputs, like we do for most of the analyses in this thesis. As we defined in Theorem 2.1.1, f is obtained by applying a Heaviside step function to $h^{(L+1)}$. This makes $P(f)$ itself not analytically computable except for a small input spaces \mathcal{X} . However it can be approximated by the EP approximation as we will see later. For brevity, in the rest of this discussion, we will omit the layer dependence, writing h for the pre-activations of the last layer.

In Poole et al. [2016], the authors showed that for i.i.d. parameters, if correlations are furthermore ignored between the hidden neurons of each layer (mean field approximation), an analytical expression for the correlation of the pre-activation output of a DNN $\mathbf{E}[h(x)h(x')]$ for two inputs x and x' could be obtained. In later work Lee et al. [2018] showed that these correlations go away in the limit that the number of hidden neurons per layer goes to infinity, so that the mean field assumption is exact in this limit. Furthermore, by studying the joint distribution of pre-activation outputs for several inputs $(h(x_1), \dots, h(x_n))$, it was shown that in this limit of infinite width, $\tilde{P}(h)$ approaches a Gaussian process (GP), which they called a Neural network Gaussian process (NNGP) [Lee et al., 2018]. Specifically, this means that the

joint distribution of $(h(x_1), \dots, h(x_n))$ is given by a multivariate Gaussian

$$\tilde{P}(h(x_1) = \tilde{y}_1, \dots, h(x_n) = \tilde{y}_n) \propto \exp\left(-\frac{1}{2}\tilde{\mathbf{y}}^T \mathbf{K}^{-1} \tilde{\mathbf{y}}\right), \quad (\text{B.3})$$

where $\tilde{\mathbf{y}} = (\tilde{y}_1, \dots, \tilde{y}_n)$. The matrix \mathbf{K} is the covariance matrix, so that its entries $\mathbf{K}_{ij} = \mathbf{E}[h(x_i)h(x_j)]$. The function giving the covariance for any pair of inputs $K(x, x') = \mathbf{E}[h(x)h(x')]$ is called the *kernel function*. The kernel function of an NNGP is determined by the architecture of the infinite-width DNN, and parameter distribution P_{par} (under the usual assumptions only the weight and bias variances σ_w^2 and σ_b^2 need to be specified).

B.2.1 Computing the kernel of NNGPs

The NNGP kernel for fully connected ReLU networks has a well known analytical form known as the arccosine kernel ([[]]Cho and Saul [2009]), while for convolutional and residual networks without pooling it can be efficiently computed[[]]¹. For more complicated architectures we use the Monte Carlo method proposed in Novak et al. [2018a] to empirically estimate the kernel $K(x, x')$. We sample M parameters $\{\theta_m\}_{m=1}^M$ i.i.d. from P_{par} , and compute the empirical estimate of the kernel as follows

$$\tilde{K}(x, x') := \frac{\sigma_w^2}{Mn} \sum_{m=1}^M \sum_{c=1}^n \sigma((h_{\theta_m}^L(x))_c) \sigma((h_{\theta_m}^{L-1}(x'))_c) + \sigma_b^2 \quad (\text{B.4})$$

where $(h_{\theta_m}^L(x))_c$ is the pre-activation of the c th neuron in the last *hidden* layer (L is the number of hidden layers) for the network with parameters θ_m at input x , n is the number of neurons in the last hidden layer, and the function σ is the non-linearity. σ_w^2 and σ_b^2 are the weight and bias variances, respectively. This can be derived from the expression, in equation 19² in Novak et al. [2018a] by applying one step of the NNGP recurrence relation (Eq 4 in [Lee et al., 2017]) to find the correlation \tilde{K} of the network's pre-activation outputs $h_{\theta}^{L+1}(x)$ from the covariance of the previous layer's activations, which simply corresponds to multiplying by σ_w^2 and adding

¹We use the code from Garriga-Alonso et al. [2018] to compute the kernel for convolutional networks for some of our experiments

²we don't have the spatial parameters α, α' because we assume that the last hidden layer is a "flat layer", and the last layer of weights if fully connected

σ_b^2 . This allows us to use every neuron in the last hidden layer as a quasi-independent estimate of $K(x, x')$ thus reducing the variance, at the expense of some bias (because they are only fully independent in the limit of infinite width).

In Novak et al. [2018a], they argue that this sampling incurs an error of order $1/\sqrt{Mn}$. In our experiments we use $M = 0.1m$ where m is the number of training examples, and n is the width of the last hidden layer defined by the architecture (see the list of architectures in Section A.2.1 and Section A.3.2). We chose this number so that Mn is comfortably larger than m for any architecture we use. Because this is (approximately) the number of independent samples of the empirical covariance, this should ensure that with high probability the empirical estimate of the covariance is full rank, which is necessary for the NNGP calculations.

B.3 Computation of posterior and marginal likelihood

In this section, we provide more details on the techniques used to compute quantities of interest for NNGPs. Specifically we will see how to compute marginal likelihood for the experiments in Chapter 5, and posteriors for experiments in Chapter 6. We assume we have a kernel matrix for the inputs of interest, computed as described in Section B.2.1. The computations are different depending on the likelihood used. In Chapter 5 we only use the 0 – 1 likelihood, while in Chapter 6 we use both 0 – 1 and Gaussian likelihoods.

B.3.1 NNGP with 0-1 likelihood and EP approximation.

For noiseless classification, the most natural likelihood is the 0 – 1 likelihood defined in Section B.1. As we mentioned before, $\tilde{P}(h)$ is a GP, while $P(f)$ is not because of the nonlinearity in the output layer. In Bayesian terminology h would be called a *latent variable* which is linked to the likelihood via a *linking function*. This is a technical requirement necessary to use some of the mathematical formalism and software packages for GPs, so we present it here only for completeness. In our case, using the 0 – 1 likelihood, and Heaviside step function connecting f and h (Section 2.1), we can decompose $P(S|h) = P(\mathbf{Y}|h)$ (omiting \mathbf{X} dependence) as $P(\mathbf{Y}|h) = \prod_i P(y_i|h(x_i))$ Because $y_i \in \{0, 1\}$, $P(y_i|h(x_i))$ is Bernoulli distribution, and its

single probability parameter is determined by $h(x_i)$ by a Heaviside linking function

$$P(y_i = 1|h(x_i)) = \begin{cases} 1 & \text{if } h(x_i) > 0 \\ 0 & \text{otherwise .} \end{cases}$$

When training the DNN with an optimizer, we cannot use the $0 - 1$ likelihood because its gradient is 0 almost everywhere. We therefore use a smooth output nonlinearity, like a sigmoid function. This results in a sigmoid or ‘Probit’ linking function, and the log-likelihood correspond to the commonly used ‘cross-entropy’ (CE) loss, which is the most commonly used loss function in classification problems. Because we train to 0 training error, even when using CE loss, we may argue that the $0 - 1$ likelihood could be a good approximation to the behaviour of our optimization procedure. Furthermore, we have found that using $0 - 1$ or sigmoid (CE) likelihood for the GP calculations gives very similar results in our experiments. Because of this we sometimes informally refer to the GP using $0 - 1$ likelihood as “using the CE loss” when comparing it with SGD training with CE loss in Chapter 6.

Unfortunately, $0 - 1$ likelihood makes the posterior of the GP analytically intractable. We therefore use a standard approximation technique known as expectation propagation (EP) [Rasmussen, 2004], which approximates the posterior over the latent function as a Gaussian, which we can sample from and then use the Heaviside function to predict the binary labels at the test points. We use this technique to approximate posterior probabilities of the GP with $0 - 1$ likelihood, by sampling. However, in most cases we estimate posterior probabilities by a ratio of likelihoods as explained next.

The EP algorithm can also be used to estimate the marginal likelihood [Rasmussen, 2004]. This gives the probability of a labelling of a set of points. Remember that in the Bayesian formalism $P(S) = P(\mathbf{Y}|\mathbf{X})$ gives the probability the input points x_i in the training set have labels y_i . We can similarly identify a function f with the event that the set of input points x in the whole domain \mathcal{X} have labels $f(x)$, which is analogous, and can thus be computed in the same manner as the marginal likelihood! Furthermore, the posterior in Equation (6.2) for a

function f which is compatible with S , can then be simply expressed as

$$P(f|S) = \frac{P(f)}{P(S)},$$

where both $P(f)$ and $P(S)$ are readily computed using the EP algorithm to approximate marginal likelihood. This is the method we use to estimate $P(f|S)$ for 0–1 likelihood, without sampling. We will refer to this method as the ‘ratio of log-likelihoods’ approximation or the log p approximation. We have found that this method gives very similar results to the estimates using sampling from the approximate posterior (see Figure B.1c).

For the LSTM experiments in Chapter 6, we used the CE likelihood, rather than the 0–1 likelihood because the EP approximation was numerically unstable with the 0–1 likelihood for this system. The smooth version is described in Section B.4, and we empirically found that the two gave very similar estimates of probability.

B.3.2 NNGP with Gaussian likelihood

For this formulation, we consider the output space to be the real numbers, $\mathcal{Y} = \mathbb{R}$. The functions are therefore real-valued and have a prior given by the infinite width NNGP limit of $\tilde{P}(h)$. The likelihood is a Gaussian likelihood defined as

$$P(S|h) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(h(x_i) - y_i)^2\right), \quad (\text{B.5})$$

where σ^2 is the error variance. S is as before so that in particular $y_i \in \{0, 1\}$.

This likelihood allows us to analytically compute the exact posterior for h [Rasmussen, 2004]. In the experiments in the paper we therefore sampled from this exact posterior, to get values of $h(x)$ at the test points, which were then thresholded at 0 to find the predicted class label. The log-likelihood in this case corresponds to the mean squared error loss, so that we train SGD with that loss when comparing with this GP likelihood in Chapter 6. Because of this we informally refer to the GP with Gaussian likelihood as ‘using MSE loss’. We have chosen a small value of the error variance $\sigma^2 = 0.002$, to simulate SGD achieving a small value of the MSE loss.

Note that under the standard assumption that training and test instances come from the same distribution, this algorithm may be considered to be not fully Bayesian in the sense that the training and test labels are treated differently (Gaussian likelihood in training points versus Bernoulli likelihood (by thresholding) at test points). Nevertheless, this is a standard technique in analyses of NNGPs, because it allows a tractable analysis.

B.4 Empirical results concerning the GP approximations

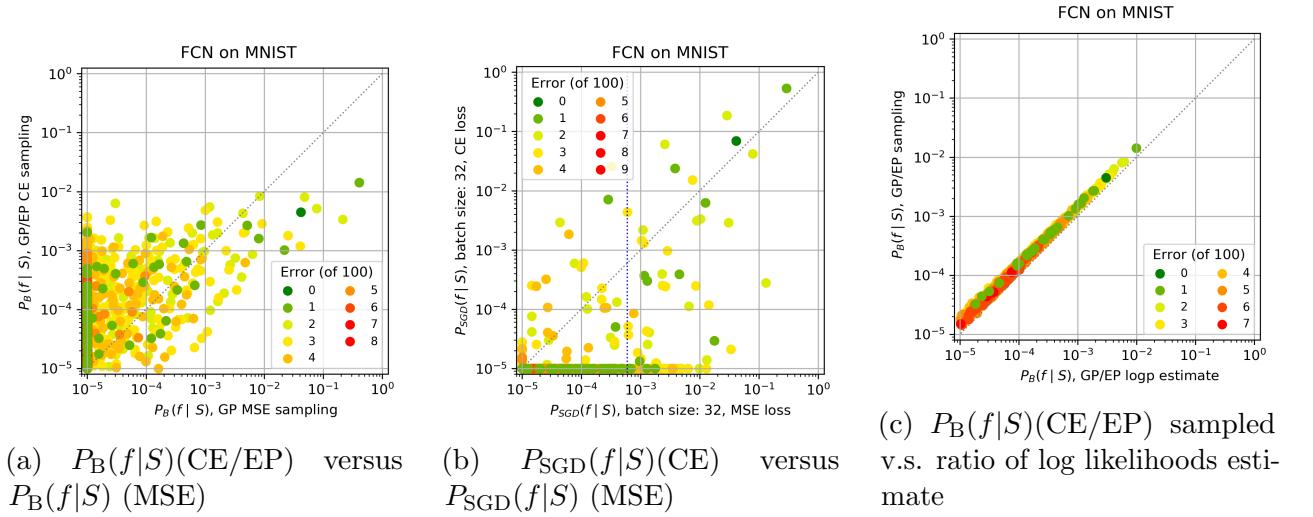


Figure B.1: Comparing GP approximations with CE and MSE loss for FCN on MNIST In (a) we compare the behaviour of the GP approximations for $P_B(f|S)$ with MSE loss to the GP/EP approximation with CE loss. We sampled from the GP MSE posterior, and the GP/EP CE posterior distribution 10^6 times, and estimated $P_B(f|S)$ for different functions (see [[methods in chapter 5]]). It is expected that the two measures should diverge somewhat due to details of the loss function on the training data, but we find that they are correlated. (b) compares $P_{\text{SGD}}(f|S)$ with MSE loss and $P_{\text{SGD}}(f|S)$ with CE loss. Functions with high $P_{\text{SGD}}(f|S)$ are scattered around $y = x$. This implies that the loss function does not substantially affect $P_{\text{SGD}}(f|S)$ on average, although it introduces some fluctuations. Note that in (a) the two methods correlate, but that 1) the GP-EP is systematically lower than the MSE, and 2) that the slope is below $x = y$. These two trends are, we believe, more general for the GP-EP approximation on CE loss, and may be due to systematic errors in the EP approximation (see Section G.2). (c) Here we compare the GP/EP $\log(p)$ approximation with GP/EP sampling. For this figure, we use only functions found by Adam in 10^6 samples, and compare probabilities found by the GP/EP $\log(p)$ approximation to those found by GP/EP sampling. We mostly use the former method, but as is clear from the above example, there is probably not much difference between them for functions with high $P_B(f|S)$.

In this section we compare the behaviour of the GP approximations (and SGD) with different loss functions / likelihoods. As detailed in Sections B.3.1 and B.3.2, there are differences in the way the GP approximation with MSE loss and the GP/EP approximation with CE loss calculate their respective estimates for $P_B(f|S)$. However, the (latent) function has the same prior in both cases, so we may expect the posterior $P_B(f|S)$ to correlate. And it is clear from Figure B.1a that they do indeed correlate. We believe that the correlation not being centred around $y = x$ is predominately due to the EP approximation because apart from scatter, the behaviour of SGD with the two loss functions is centred around $y = x$. The experiments in Section B.4.1 suggest that the errors introduced by EP may be approximately constant in log space. Our results with the EP approximation suggest that it may in fact be linear in log scale, introducing the change in slope we observe for the EP-approximated $P_B(f|S)$ vs $P_{\text{SGD}}(f|S)$ or $P_B(f|S)$ with MSE likelihood. In Section G.2, we also compare the EP approximation with a estimation of $P_B(f|S)$ via direct sampling (and thus with controlled error) of the posterior probabilities for 0-1 likelihood for small Boolean function datasets, where these computations are feasible. We indeed find that EP tends to systematically underestimate posterior probabilities, specially for complex target functions. Overall what we find is that the EP approximation does reasonably well on relative probabilities, but less well on absolute probabilities.

To mitigate the effect of the EP approximation underestimating the probabilities, we perform a simple empirical regularisation. For systems where we find that $\sum P_B(f|S) \approx 1$ for the MSE approximation, we renormalise the $P_B(f|S)$ from the EP approximation by a constant factor such that $\sum P_B(f|S) = 1$ ³. For most systems we study the effect of this regularisation procedure is small on a log scale. This method facilitates the comparison with $P_{\text{SGD}}(f|S)$ because the potential systematic errors in the absolute values are regularised in the same way for all systems. This regularisation is applied to all experiments (for ease of comparison), unless otherwise specified. But one should be careful, because we have not proven that the only source of systematic error is the EP approximation, so we still show the raw data in some cases, and specify the amount by which the regularization has changed the raw data in all cases. Specifically, for Figure G.11 the renormalisation constant was calculated for Adam without

³Note also that because we sample to obtain $P_{\text{SGD}}(f|S)$ its empirical frequencies automatically sum to 1.

overtraining (as it had the highest raw value for $\sum P_B(f|S)$), and the probabilities in the other plots with FCN on MNIST are all adjusted by the multiplicative constant of 3.59, which is modest on the full log scale of the graphs. The two systems for which this renormalisation has a larger effect are the LSTM and the ionosphere dataset. While for both systems the MSE sampling looks relatively close to $y = x$, the raw EP approximation has significantly lower probabilities. The renormalisation factors were 1.15×10^5 and 97 respectively. We are currently running larger MSE experiments to independently check the correlation for these two systems.

In Section B.3.1, we described using a Heaviside linking function when using the 0-1 loss function in the EP approximation. To compare this with the CE likelihood we have also sampled f following a Bernoulli distribution with a Probit linking function. To test the differences between the results (which we assume to be small), we tested 100 randomly selected functions (on MNIST) with $\langle \epsilon_G \rangle$ ranging from 0% to 100%. Of these, the average difference between the results as a percentage of the magnitude of the log probabilities was 0.013%, and the maximum was 0.58% for the FCN architecture. We also compare the GP/EP sampling with GP/EP $\log p$ approximation in Figure B.1c (see Section B.3.1), and find that they give very similar results.

Most results in Chapter 6 use the GP/EP $\log p$ approximation rather than sampling. We use the 0 – 1 loss because it should approximate better the behaviour of SGD trained with CE, and we used the $\log p$ method because the EP probabilities are often significantly underestimated (an extreme example is the LSTM system), so that sampling from the approximate posterior would require too many samples to obtain good estiamtes of the probabilities. The limitation of this approach is that we could technically miss some functions which have high probability with GP and CE likelihood, but too low probability for SGD to appear in the sample. We rely on the results in this section that GP/EP sampling (CE likelihood) correlates well with sampling from GP with MSE likelihood, and there are no outliers. If this is true of all the systems we have explored, then the MSE results in Chapter 6 would imply that there are no outliers we missed in th experiments with GP and CE.

B.4.1 Comparing approximate GP marginal likelihoods with direct sampling for MNIST

In the previous experiments, we have compared several approximations to GP probabilities based on different likelihoods, showing that they give similar results, and increasing our confidence that the approximations give reasonable answers. Furthermore, in Section G.2, compare the GP approximations to posteriors with direct sampling with 0 – 1 likelihood for a Boolean system with a small input space, which offers a gold standard to compare against. Here we offer a complementary experiment, where we restrict the input space to be small, but the inputs are from MNIST, offering a more realistic situation than the Boolean system.

In Figure B.2, we show results comparing the empirical frequency of labellings for a sample of 10 random (but fixed) MNIST images, when these frequencies are obtained by sampling parameters of a neural network (with a Gaussian distribution with parameters $\sigma_w = \sigma_b = 1.0$), versus the probability calculated using two methods to approximate the marginal likelihood of the Gaussian process corresponding to the neural network architecture we use. We compare the Laplacian and EP approximations.(see Rasmussen [2004] for a description of the algorithms) The network has 2 fully connected hidden layers of 784 ReLU neurons each. We find that there is a good correlation between the approximate and sampled probabilities, but the EP probabilities tend to be systematically underestimated for most labellings. In larger sets of inputs (1000), we also found that the relative difference between the log-likelihoods given by the two approximations was less than about 10%, which suggests that perhaps the errors introduced by these approximations are constant in log space.

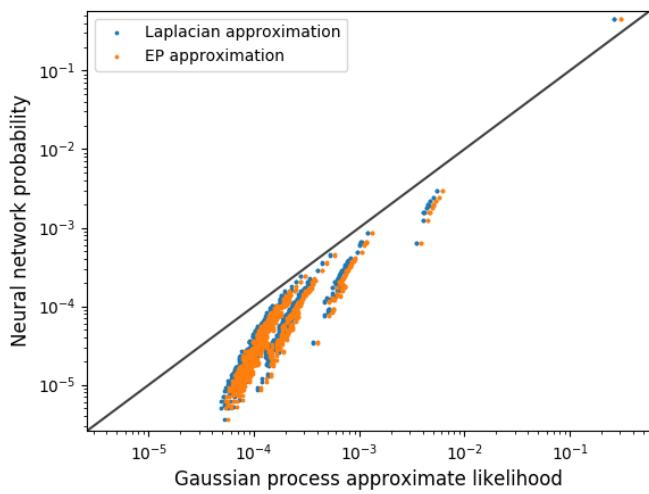


Figure B.2: Comparing the empirical frequency of different labellings for a sample of 10 MNIST images obtained from randomly sampling parameters from a neural neural network, versus the approximate marginal likelihood from the corresponding Gaussian process. Orange dots correspond to the expectation-propagation approximation, and blue dots to the Laplace approximation. The network has 2 fully connected hidden layers of 784 ReLU neurons each. The weight and bias variances are 1.0.

Appendix C

Proofs

C.1 Proof of uniformity

This proof was done by the author, David Martinez Rubio, and Chris Mingard.

For convenience we repeat some notation we use in this section. Let $\{0, 1\}^n$ be the set of vertices of the n -dimensional hypercube. We use $\langle \cdot, \cdot \rangle$ to refer to the standard inner product in \mathbb{R}^n . Define the function $\mathcal{T} : \mathbb{R}^n \rightarrow \mathbb{N}$ as the number of vertices of the hypercube that are above the hyperplane with normal vector w and that passes through the origin. Formally $\mathcal{T}(w) = |\{x \in \{0, 1\}^n : \langle w, x \rangle > 0\}|$. We use \odot for element-wise multiplication of two vectors.

We slightly abuse notation and denote the probability density function corresponding to a probability measure P , with the same symbol, P . The arguments of the function or context should make clear which one is meant.

Proof strategy. We consider the sampling of the normal vector w as a two-step process: we first sample the absolute values of the elements, giving us a vector w_{pos} with positive elements¹, and then we sample the signs of the elements. Our assumption on the probability distribution implies that each of the 2^n sign assignments is equally probable, each happening with a probability 2^{-n} . The key of the proof is to show that for any w_{pos} , each of the sign assignments gives a distinct value of T (and because there are 2^n possible sign assignments, for any value of T , there is exactly one sign assignment resulting in a normal vector with that value of T). This implies that, provided all sign assignments of any w_{pos} are equally likely, the

¹almost surely, assuming 0 has zero probability measure

distribution on T is uniform.

Theorem 3.3.1 *Let P be a probability measure on \mathbb{R}^n , which is symmetric under reflections along the coordinate axes, so that $P(x) = P(Rx)$, where R is a reflection matrix (a diagonal matrix with elements in $\{-1, 1\}$). Let the weights of a perceptron without bias, w , be distributed according to P . Then $P(T = t)$ is the uniform measure.*

Before proving the theorem, we first need a definition and a lemma.

Definition We define the function mapping a vector from $\{-1, 1\}^n$ (which we interpret as the signature of the weight vector), and a vector of nonnegative reals (which we interpret as the absolute values of the elements of the weight vector) to the value of t of the corresponding weight vector:

$$\begin{aligned} K : \{-1, 1\}^n \times \mathbb{R}_{\geq 0}^n &\rightarrow \{0, 1, \dots, 2^n - 1\} \\ (\sigma, a) &\mapsto \mathcal{T}(\sigma \odot a) \end{aligned} \tag{C.1}$$

Lemma C.1.1. *The function K is bijective with respect to its first argument, for any value of its second argument except for a set of measure 0.*

Proof of Theorem C.1.1. Because the cardinality of the codomain of K is the same as the domain of its first argument, it is enough to prove injectivity of K with respect to its first argument.

Fix $a \in \mathbb{R}_{\geq 0}^n$ satisfying that the following set has cardinality 3^n :

$$\tilde{\mathbb{S}}_a = \{\langle x, a \rangle : x \in \{-1, 0, 1\}^n\}.$$

Note that the set of a in which some pair of elements in the definition of \mathbb{S}_a is equal has measure zero, because their equality implies that a lies within a hyperplane in \mathbb{R}^n . Let us also define the set of subsums of elements of a :

$$\mathbb{S}_a = \{\langle x, a \rangle : x \in \{0, 1\}^n\}.$$

which has cardinality 2^n for the a considered.

Now, consider a natural bijection $J_a : \{-1, 1\}^n \rightarrow \mathbb{S}_a$ induced by the bijective mapping of signatures $\sigma \in \{-1, 1\}^n$ to vertices of the hypercube $\{0, 1\}^n$ by mapping -1 to 0 and 1 to 1 . To be more precise, $J_a(\sigma) = \sum_{i=1}^n a_i \frac{(\sigma(i)+1)}{2}$.

Then, we claim that

$$K(\sigma, a) = |\{s \in \mathbb{S}_a : s < J_a(\sigma)\}| \quad (\text{C.2})$$

This implies that for the a we have fixed K is injective, for if two σ mapped to the same value, their corresponding value of $J_a(\sigma)$ should be the same, giving a contradiction. So it only remains to prove equation C.2.

Let us also first denote, for $x \in \{0, 1\}^n$ and $s, s' \in \mathbb{S}_a$

$$\begin{aligned} \Sigma(x) &:= \langle x, a \rangle \\ s \cap s' &:= \langle (\Sigma^{-1}(s) \cap \Sigma^{-1}(s')), a \rangle \\ s \cup s' &:= \langle (\Sigma^{-1}(s) \cup \Sigma^{-1}(s')), a \rangle \\ \bar{s} &:= \langle (\overline{\Sigma^{-1}(s)}), a \rangle \end{aligned}$$

where we interpret elements of $\{0, 1\}^n$ as subsets of $\{1, \dots, n\}$. The notation above lets us interpret subsums in S_a as subsets of entries of a . Note that Σ^{-1} is well defined for the fixed a we are considering.

Now, let $\sigma \in \{-1, 1\}^n$, $s' = J_a(\sigma)$, and consider an $s \in \mathbb{S}_a$ such that $s < s'$. Then $s \cap s' + s \cap \bar{s}' = s < s'$ so

$$-s \cap s' + s' - s \cap \bar{s}' > 0 \quad (\text{C.3})$$

Now, let the operation $*$ (we omit dependence on s') be defined for any $u \in \mathbb{S}_a$ as $u^* := u \cap \bar{s}' + (s' - u \cap s') \in \mathbb{S}_a$. Since $s' = J_a(\sigma)$ we have

$$\langle (\sigma \odot a), \Sigma^{-1}(s^*) \rangle = -s \cap \bar{s}' + (s' - s \cap s').$$

Using equation C.3,

$$\langle (\sigma \odot a), \Sigma^{-1}(s^*) \rangle > 0 \iff s < s'.$$

Therefore, all the points $\Sigma^{-1}(s^*)$ for $s < s'$ are above the hyperplane with normal $(\sigma \odot a)$, and

all points $\Sigma^{-1}(s^*)$ for $s \geq s'$ are below or precisely on the hyperplane. All that is left is to show the converse, all points which are above the hyperplane are $\Sigma^{-1}(s^*)$ for one and only one $s < s'$. It suffices to show that the operation $*$ is injective for all s (as bijectivity follows from the domain and codomain being the same). By contradiction, let s and u map to the same value under $*$, then $s \cap s' - s \cap \bar{s}' = u \cap s' - u \cap \bar{s}'$, which implies $s \cap s' = u \cap s'$ and $s \cap \bar{s}' = u \cap \bar{s}'$, for the a we are considering, and so $s = u$. Therefore $*$ is injective, and equation C.2 follows. \square

Proof of Theorem 3.3.1.

$$P(T = t') = P(w : \mathcal{T}(w) = t') = \int \mathbf{1}_{\mathcal{T}(w)=t'} P(w) d^n w$$

Now, we can divide the integral into the quadrants corresponding to different signatures of w , and we can let $P(w) = \frac{1}{2^n} \tilde{P}(|w|)$, because it is symmetric under reflections of the coordinate axes.

$$\begin{aligned} P(w : \mathcal{T}(w) = t') &= \sum_{\sigma \in \{-1,1\}^n} \int_{\mathbb{R}_{\geq 0}^n} \frac{1}{2^n} \tilde{P}(a) d^n a \mathbf{1}_{\mathcal{T}(\sigma \odot a)=t'} \\ &= \frac{1}{2^n} \int_{\mathbb{R}_{\geq 0}^n} \tilde{P}(a) d^n a \sum_{\sigma \in \{-1,1\}^n} \mathbf{1}_{\mathcal{T}(\sigma \odot a)=t'} \\ &= \frac{1}{2^n} \int_{\mathbb{R}_{\geq 0}^n} \tilde{P}(a) d^n a \cdot 1 \\ &= \frac{1}{2^n}. \end{aligned}$$

The third equality follows from Theorem C.1.1. Indeed, bijectivity implies that for any a , except for a set of measure 0, there is one and only one signature which results in t' .

\square

C.2 $P(t = 0)$ for perceptron with infinitesimal b

This argument was mainly due to Chris Mingard

If b is sampled uniformly from $[-\epsilon, \epsilon]$, then only if $|\langle w, x \rangle| < \epsilon$ can some x be classified differently from a perceptron without a threshold bias term. The set of weight vectors which

change the classification of non-zero x becomes vanishingly small as ϵ goes to 0, but for $x = 0$, we have $P(\mathbf{1}(\langle w, 0 \rangle + b) = 0) = P(\mathbf{1}(\langle w, 0 \rangle + b) = 1) = 1/2$. Consider some function f , and define g where $f \rightarrow g$ under the addition of an infinitesimal bias. Then with even probability the origin remains mapped to 0 (meaning $\mathcal{T}(g) = \mathcal{T}(f)$), or is mapped to 1 (meaning $\mathcal{T}(g) = \mathcal{T}(f) + 1$) as the rest of f is unchanged, to $\mathcal{O}(\epsilon)$. As this is true of all f , $P(T = t)_{b \sim (-\epsilon, \epsilon)} = \frac{1}{2}P(T = t) + \frac{1}{2}P(T = t - 1)$, leading to:

$$P(T = t) = \begin{cases} 2^{-(n+1)} & \text{if } t = 0 \text{ or } t = 2^n \\ 2^{-n} & \text{otherwise} \end{cases}. \quad (\text{C.4})$$

For larger σ_b $P(t = 0)$ or $P(t = 2^n)$ increases with increasing σ_b as can be seen in Figure 3.3 in Chapter 3.

C.3 Bounding Boolean function complexity, K_{Bool} , with t

This proof was primarily the work of Joar Skalse.

Theorem C.3.1. $n \times \min(t, 2^n - t) - 1$ is an upper bound on the complexity of Boolean functions f for which $\mathcal{T}(f) = t$.

Proof. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a function s.t. $\mathcal{T}(f) = t$.

Let $x_1 \dots x_n$ be propositional variables and let each assignment to $x_1 \dots x_n$ correspond to a vector in $\{0, 1\}^n$ in the straightforward way.

Let ϕ be the Boolean formula $\bigvee \{\bigwedge \{\text{if } v_i = 1 \text{ then } x_i \text{ else } \neg x_i \mid v_i \in v\} \mid v \in \{0, 1\}^n, f(v) = 1\}$. The formula ϕ expresses f as a Boolean formula in Disjunctive Normal Form (DNF).

Let ψ be the Boolean formula $\bigwedge \{\bigvee \{\text{if } v_i = 0 \text{ then } x_i \text{ else } \neg x_i \mid v_i \in v\} \mid v \in \{0, 1\}^n, f(v) = 0\}$. The formula ψ expresses f as a Boolean formula in Conjunctive Normal Form (CNF).

Since f maps t out of the 2^n vectors in $\{0, 1\}^n$ to 1 it must be the case that ϕ has t clauses and ψ has $2^n - t$ clauses. Each clause contains $n - 1$ binary connectives, and there is one connective between each clause. Hence ϕ contains $n \times t - 1$ binary connectives and ψ contains

$n \times (2^n - t) - 1$ binary connectives. Therefore f is expressed by some Boolean formula of complexity $n \times \min(t, 2^n - t) - 1$.

Since f was chosen arbitrarily, if a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ maps t inputs to 1 then the complexity of f is at most $n \times \min(t, 2^n - t) - 1$. \square

Theorem C.3.2. *Let C be a defined recursively as follows;*

$$C(n, 0) = 0$$

$$C(n, 2^n) = 0$$

$$C(n, 1) = n - 1$$

$$C(n, 2^n - 1) = n - 1$$

$$C(n, t) = C(n - 1, \lceil t/2 \rceil) + C(n - 1, \lfloor t/2 \rfloor) + 2$$

Then $C(n, t)$ is an upper bound on the complexity of Boolean functions f for which $\mathcal{T}(f) = t$.

Proof. Let $P(n)$ be that $C(n, t)$ is an upper bound on the complexity of Boolean functions f over n variables s.t. $\mathcal{T}(f) = t$.

Base case $P(1)$: If ϕ is a Boolean formula defined over 1 variable then ϕ is equivalent to True, False, x_1 , or $\neg x_1$. We can see by exhaustive enumeration that $P(1)$ holds in each of these four cases.

Inductive step $P(n) \rightarrow P(n + 1)$: Let ϕ be a Boolean formula defined over $n + 1$ variables s.t. $\mathcal{T}(\phi) = t$.

Case 1. $t = 0$: If $t = 0$ then $\phi \equiv \text{False}$, and so the complexity of ϕ is 0. Hence the inductive step holds.

Case 2. $t = 2^n$: If $t = 2^n$ then $\phi \equiv \text{True}$, and so the complexity of ϕ is 0. Hence the inductive step holds.

Case 3. $t = 1$: If $t = 1$ then ϕ has just a single satisfying assignment. If this is the case then ϕ can be expressed as a formula of length n written in Disjunctive Normal Form, and hence the inductive step holds.

Case 4. $t = 2^n - 1$: If $t = 2^n - 1$ then ϕ has just a single non-satisfying assignment. If this is the case then ϕ can be expressed as a formula of length n written in Conjunctive Normal Form, and hence the inductive step holds.

Case 5. $1 < t$ and $t < 2^n - 1$: If ϕ is a Boolean formula defined over $n + 1$ variables then ϕ is

logically equivalent to a formula $(x_{n+1} \wedge \psi_1) \vee (\neg x_{n+1} \wedge \psi_2)$, where ψ_1 and ψ_2 are defined over $x_1 \dots x_n$. Let t_1 be the number of assignments to $x_1 \dots x_n$ that are mapped to 1 by ψ_1 , and let t_2 be the corresponding value for ψ_2 .

By the inductive assumption the complexity of ψ_1 and ψ_2 is bounded by $C(n, t_1)$ and $C(n, t_2)$ respectively. Therefore, since $\phi \equiv (x_{n+1} \wedge \psi_1) \vee (\neg x_{n+1} \wedge \psi_2)$ it follows that the complexity of ϕ is at most $C(n, t_1) + C(n, t_2) + 2$. Since $t_1 + t_2 = t$, and since $C(n, t_a) < C(n, t_b)$ if t_b is closer to 2^{n-1} than t_a is (lemma C.3.3), it follows that the complexity of ϕ is bounded by $C(n, t) = C(n - 1, \lceil t/2 \rceil) + C(n - 1, \lfloor t/2 \rfloor) + 2$.

Since Case 1 to 5 are exhaustive the inductive step holds. \square

Lemma C.3.3. *If $t + 1 \leq 2^{n-1}$ then $C(n, t) < C(n, t + 1)$.*

Proof. Let $P(n)$ be that if $t + 1 \leq 2^{n-1}$ then $C(n, t) < C(n, t + 1)$.

Base case $P(2)$: We can see that $C(4, 0) = 1$, $C(4, 1) = 2$, $C(4, 2) = 4$, $C(4, 3) = 2$ and $C(4, 4) = 1$. By exhaustive enumeration we can see that $P(2)$ holds.

Inductive step $P(n) \rightarrow P(n + 1)$:

Case 1. *t is even:*

$$\begin{aligned} C(n + 1, t) - C(n + 1, t + 1) &= (C(n, t/2) + C(n, t/2) + 2) \\ &\quad - (C(n, t/2) + C(n, t/2 + 1) + 2) \\ &= C(n, t/2) - C(n, t/2 + 1) \end{aligned}$$

If t is even and $t + 1 \leq 2^{(n+1)-1}$ then $t/2 + 1 \leq 2^{n-1}$. Hence $C(n, t/2) - C(n, t/2 + 1) < 0$ by the inductive assumption, and so $C(n + 1, t) - C(n + 1, t + 1) < 0$.

Case 2. *t is odd:*

$$\begin{aligned} C(n + 1, t) - C(n + 1, t + 1) &= (C(n, (t + 1)/2) + C(n, (t - 1)/2) + 2) \\ &\quad - (C(n, (t + 1)/2) + C(n, (t + 1)/2) + 2) \\ &= C(n, (t - 1)/2) - C(n, (t + 1)/2) \end{aligned}$$

If t is odd and $t + 1 \leq 2^{(n+1)-1}$ then $(t + 1)/2 \leq 2^{n-1}$. Hence $C(n, (t - 1)/2) - C(n, (t + 1)/2) < 0$ by the inductive assumption, and so $C(n + 1, t) - C(n + 1, t + 1) < 0$.

Since Case 1 and 2 are exhaustive the inductive step holds. \square

C.4 Theorems associated with entropy increase for DNNs

The following results were primarily the work of the author, and Chris Mingard.

We define the data matrix for a general set of input points below.

Definition C.4.1 (Data matrix). *For a general set of inputs, $\{x^{(i)}\}_{i=1,\dots,m}$, $x_i \in \mathbb{R}^n$, we define the data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ which has elements $X_{ij} = x_j^{(i)}$, the j -th component of the i -th point.*

Theorem 3.4.3. *For any set of inputs \mathbb{S} , the probability distribution on T of a fully connected feedforward neural network with linear activations, no bias, and i.i.d. initialisation of the weights is equivalent to an perceptron with no bias and i.i.d. weights.*

Proof. Consider a neural network with L layers and weight matrices $w_0 \dots w_L$ (notation in Section 3.4) acting on the set of points \mathbb{S} . The output of the network on an input point $x \in \mathbb{S}$ equals $\tilde{w}x$ where $\tilde{w} = w_L w_{L-1} \dots w_1 w_0$. As the weight matrices w_i are i.i.d., their distributions are spherically symmetric $P(w_i = aR) = P(w_i = a)$ for any rotation matrix R in $\mathbb{R}^{n_i \times n_i}$ and matrix $a \in \mathbb{R}^{n_{i+1} \times n_i}$. This implies that $P(\tilde{w} = aR) = P(\tilde{w} = a)$. Because the value of T is independent of the magnitude of the weight, $|\tilde{w}|$, this means that $P(T = t)$ is equivalent to that of an perceptron with i.i.d. (and thus spherically symmetric) weights. \square

One can make Theorem 3.4.3 stronger, by only requiring the first layer weights w_0 to have a spherically symmetric distribution, and be independent of the rest of the weights. If the set of inputs is the hypercube, $\mathbb{S} = \mathcal{H}^n$, one needs even weaker conditions, namely that the distribution of w_0 is symmetric under reflections along the coordinate axes (as in Theorem 3.3.1) and has signs independent of the rest of the layer's weights. This implies that the condition of Theorem 3.3.1 is satisfied by \tilde{w} .

Theorem 3.4.4. *Applying a ReLU function in between each layer produces a lower bound on $P(T = 0)$ such that $P(T = 0) \geq 2^{-n}$.*

Proof. Consider the action of a neural network $\langle n, l_1, \dots, l_p, 1 \rangle$ with ReLU activation functions on $\{0, 1\}^n$. After passing $\{0, 1\}^n$ through $l_1 \dots l_p$ and applying the final ReLU function after l_p ,

all points must lie in $\mathbb{R}_{\geq 0}^n$ by the definition of the ReLU function. Then, if w is sampled from a distribution symmetric under reflection in coordinate planes:

$$2^{-n} = P(w \cdot \mathbb{R}_{\geq 0}^n \leq 0) \leq P(T = 0, \text{ReLU})$$

This result also follows from Theorem C.4.7.

We observe $P(T = 2^n - 1) \approx P(T = 0)$ because the two states are symmetric except in cases where multiple points are mapped to the origin. This happens with zero probability for infinite width hidden layers. \square

Theorem 3.4.5 *Let \mathbb{S} be a set of $m = |\mathbb{S}|$ input points in \mathbb{R}^n . Consider neural networks with i.i.d. Gaussian weights with variances σ_w^2/\sqrt{n} and biases with variance σ_b , in the limit where the width of all hidden layers n goes to infinity. Let $N1$ and $N2$ be such a neural networks with L and $L + 1$ infinitely wide hidden layers, respectively, and no bias. Then, the following holds: $\langle H(T) \rangle$ is smaller than or equal for $N2$ than for $N1$. It is strictly smaller if there exist pairs of points in \mathbb{S} with correlations less than 1. If the networks has sufficiently large bias ($\sigma_b > 1$ is a sufficient condition), the result still holds. For smaller bias, the result holds only for sufficiently large number of layers L .*

Proof. The covariance matrix of the activations of the last hidden layer of a fully connected neural network in the limit of infinite width has been calculated and are given by the following recurrence relation for the covariance of the outputs² at layer l [[cite]]:

$$K^l(x, x') = \sigma_b^2 + \frac{\sigma_w^2}{2\pi} \sqrt{K^{l-1}(x, x)K^{l-1}(x', x')} (\sin \theta_{x,x'}^{l-1} + (\pi - \theta_{x,x'}^{l-1}) \cos \theta_{x,x'}^{l-1}) \quad (\text{C.5})$$

$$\theta_{x,x'}^l = \cos^{-1} \left(\frac{K^l(x, x')}{\sqrt{K^l(x, x)K^l(x', x')}} \right) \quad (\text{C.6})$$

²Note that we can speak interchangeably about the correlations at hidden layer $l - 1$, or the correlations of the output at layer l , as the two are the same. This is a standard result, which we state, for example, in the proof of Theorem C.4.7

For the variance the equation simplifies to

$$K^l(x, x) = \sigma_b^2 + \frac{\sigma_w^2}{2} K^{l-1}(x, x) \quad (\text{C.7})$$

The correlation at layer l is $\rho^l(x, x') = \frac{K^l(x, x')}{\sqrt{K^l(x, x)K^l(x', x')}}$, can be obtained using Equation (C.5) above recursively as

$$\rho^l(x, x') = \frac{\sigma_b^2 + \frac{\sigma_w^2}{2} \sqrt{K^{l-1}(x, x)K^{l-1}(x', x')}}{\sqrt{\sigma_b^2 + \frac{\sigma_w^2}{2} K^{l-1}(x, x)} \sqrt{\sigma_b^2 + \frac{\sigma_w^2}{2} K^{l-1}(x', x')}} \rho_0^l(x, x'),$$

where

$$\rho_0^l(x, x') = \frac{1}{\pi} \left(\sin \cos^{-1}(\rho^{l-1}(x, x')) + (\pi - \cos^{-1}(\rho^{l-1}(x, x'))) \rho^{l-1}(x, x') \right).$$

For the case when $\sigma_b = 0$, this simply becomes to $\rho^l(x, x') = \rho_0^l(x, x')$.

This function is 1 when $\rho^{l-1}(x, x') = 1$, and has a positive derivative less than 1 for $0 \leq \rho^{l-1}(x, x') < 1$, which implies that it is greater than $\rho^{l-1}(x, x')$. Therefore the correlation between any pair of points increases if $\rho^{l-1}(x, x') < 1$ or stays the same if $\rho^{l-1}(x, x') = 1$, as you add one hidden layer. By Theorem C.4.2, this then implies the theorem, for the case of no bias.

When $\sigma_b > 0$, we can write, after some algebraic manipulation

$$\frac{\rho^l(x, x')}{\rho^{l-1}(x, x')} = \frac{1 + \gamma \frac{\rho_0^l(x, x')}{\rho_0^{l-1}(x, x')}}{\sqrt{1 + \gamma \left(\frac{1}{K^{l-1}(x, x)} + \frac{1}{K^{l-1}(x', x')} \right)} + \gamma^2} \quad (\text{C.8})$$

$$\geq \frac{1 + \gamma a}{\sqrt{1 + \gamma \frac{2}{\sigma_b^2} + \gamma^2}} \quad (\text{C.9})$$

$$:= \phi(\gamma), \quad (\text{C.10})$$

where $\gamma = \frac{\sigma_w^2 K^{l-1}(x, x')}{2\sigma_b^2} > 0$ and $a = \frac{\rho_0^l(x, x')}{\rho_0^{l-1}(x, x')} > 1$, and the inequality follows from $K^l(x, x) \geq \sigma_b^2$ for any x , which follows from Equation (C.5).

We will study the behaviour of $\phi(\gamma)$ as a function of γ for different values of a and σ_b^2 . For

$\gamma = 0$, this function equals 1. If $\frac{1}{a} < \sigma_b^2 < a$, its derivative is positive, and therefore is greater than 1 for $\gamma > 0$. If $a < \sigma_b^2$, there is a unique maximum for $\gamma > 0$ at $\gamma = \frac{a\sigma_b^2 - 1}{\sigma_b^2 - a}$. Because the function tends to a as $\gamma \rightarrow \infty$, if it went below 1, then at some $\gamma > 0$ it should cross 1 (by the intermediate value theorem), and by the mean value theorem, therefore it would have an extremum below one, and thus a local minimum, giving a contradiction. Thus the function is always greater than 1 when $\sigma_b > \frac{1}{a}$.

When $\sigma_b < \frac{1}{a}$, $\phi(\gamma)$ can be less than 1, thus the decreasing the correlations, for some values of γ . We know that if $\gamma \geq \frac{2(1-a\sigma_b^2)}{(a^2-1)\sigma_b^2}$ the function is greater than or equal to 1. However, because of the inequality in Equation (C.8), we can't say what happens when γ is smaller than this.

From Equation (C.7), we know that if $\sigma_w^2 \geq 2$, $K^{l-1}(x, x)$ and $K^{l-1}(x', x')$ grow unboundedly as l grows. By the above arguments applied to the expression in Equation (C.8) before taking the inequality, this implies that after some sufficiently large l , $\frac{\rho^l(x, x')}{\rho^{l-1}(x, x')}$ will be > 1 . If $\sigma_w^2 < 2$, $K^{l-1}(x, x)$ and $K^{l-1}(x', x')$ tend to $\frac{\sigma_b^2}{1-\sigma_w^2/2}$ which also becomes the fixed point of the equation for $K^l(x, x')$, Equation (C.5), implying that $\rho^l(x, x') \rightarrow 1$ as $l \rightarrow \infty$, so that the correlation must increase with layers after a sufficient number of layers, and thus the moments by Theorem C.4.2.

Finally, applying Theorem C.4.8, the theorem follows. □

Lemma C.4.2. *Consider two sets of m input points to an n -dimensional perceptron without bias, \mathcal{U} and \mathcal{V} with data matrices U and V respectively (see Theorem C.4.1). If for all $i, j = 1, \dots, m$, $(VV^T)_{ij}/\sqrt{(VV^T)_{ii}(VV^T)_{jj}} \geq (UU^T)_{ij}/\sqrt{(UU^T)_{ii}(UU^T)_{jj}}$, then every moment $\langle t^q \rangle$ of the distribution $P(T = t)$ is greater for the set of points \mathcal{V} than the set of points \mathcal{U} .*

Proof of Theorem C.4.2. We write T as:

$$T_s = \sum_{i=1}^m \mathbf{1}(\langle w, s_i \rangle)$$

$$T_u = \sum_{i=1}^m \mathbf{1}(\langle w, u_i \rangle)$$

The moments of the distribution $P(T_s = t)$ can be calculated by the following integral:

$$\langle t^q \rangle_{\mathbb{S}} = \int_S \sum_{i=0}^m \cdots \sum_{q=0}^m \mathbf{1}(\langle w, s_i \rangle) \times \cdots \times \mathbf{1}(\langle w, s_q \rangle) P(S) dS$$

Where $S = (S_1, \dots, S_q)$. Taking the sum outside the integral,

$$= \sum_{i \dots q} \int \mathbf{1}(\langle w, s_i \rangle) \times \cdots \times \mathbf{1}(\langle w, s_q \rangle) P(S) dS = \sum_{i \dots q} P(\langle w, s_i \rangle > 0, \dots, \langle w, s_q \rangle > 0)$$

The distribution for \mathcal{U} is of the equivalent form. From corollary Theorem C.4.7, we have that $P(\langle w, s_i \rangle > 0, \dots, \langle w, s_q \rangle > 0) \leq P(\langle w, u_i \rangle > 0, \dots, \langle w, u_q \rangle > 0)$. Thus we have Equation (C.11) for all q .

$$\langle t^q \rangle_{\mathbb{S}} < \langle t^q \rangle_{\mathcal{U}} \quad (\text{C.11})$$

□

Lemma C.4.3. Consider an 2-dimensional Gaussian random variable with mean μ and covariance Σ . If the correlation $\Sigma_{ij}/\sqrt{\Sigma_{ii}\Sigma_{jj}}$ increases, then

$$P(x_i > 0, x_j > 0)$$

increases

Proof. We can write the non-centered orthant probability as a Gaussian integral

$$P(x_i > 0, x_j > 0) = \int_{\mathbb{R}_{\geq 0}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} dx$$

Without loss of generality, we consider $\Sigma_{ii} = \Sigma_{jj} = 1$. Otherwise, we can rescale the variables x and obtain a new mean vector.

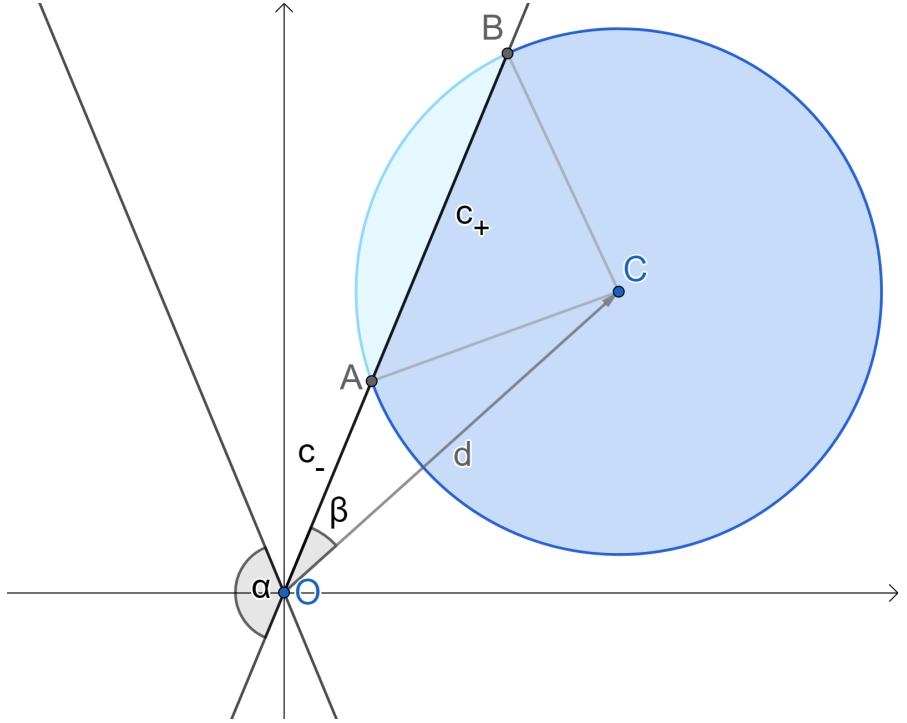


Figure C.1: Transformed 2D Gaussian for proof in Theorem C.4.3.

The covariance matrix Σ thus has eigenvalues $\lambda_+ = 1 + \Sigma_{ij}$ and $\lambda_- = 1 - \Sigma_{ij}$ with corresponding eigenvectors $(1, 1)$ and $(1, -1)$. We can rotate the axis so that $(1, 1)$ becomes $(1, 0)$. We can then rescale the x axis by $1/\sqrt{\lambda_+}$ and the y axis by $1/\sqrt{\lambda_-}$. The positive orthant becomes a cone \mathcal{C} centered around the origin and with opening angle α given by

$$\tan \alpha = \sqrt{\frac{\lambda_+}{\lambda_-}},$$

which increases when Σ_{ij} increases. The integral in polar coordinates becomes

$$\int_{\mathcal{C}} e^{-\frac{r^2}{2}} r d\theta dr.$$

Therefore, all that's left to show is that the range of θ for any r increases when α increases. See Figure C.1 for the illustration. Call γ the angle between one boundary of the cone \mathcal{C} and the position vector of the center of the Gaussian in the transformed coordinates. We can find the length of the chord between the two points of intersection between the circle and the boundary of the cone as the difference between the distances c_- , c_+ , of the segments OA and

OB, respectively. Using the cosine angle formula, we find $c_{\pm} = d \cos \gamma \pm \sqrt{r^2 - d^2 \sin^2 \gamma}$, and the chord length is $\sqrt{r^2 - d^2 \sin^2 \gamma}$, which decreases as γ increases. Furthermore, γ increases as α increases. Using the same argument for the other boundary of the cone, concludes the proof.

□

The following lemma shows that for a vector of n Gaussian random variables, the probability of two variables being simultaneously greater than 0, given that all other any fixed signs, increases if their correlation increases.

Lemma C.4.4. *Consider an n -dimensional Gaussian random variable with mean 0 and covariance Σ , $x \sim \mathcal{N}(0, \Sigma)$. Consider, for any $\sigma \in \{-1, 1\}^{n-2}$, the following probability*

$$P_{00}^{ij} := P(x_i > 0, x_j > 0 | \forall k \notin \{i, j\} \sigma_k x_k > 0)$$

If $\Sigma_{ij} / \sqrt{\Sigma_{ii}\Sigma_{jj}}$ increases, then P_{00}^{ij} increases.

Proof. We can write $P_{00}^{ij} = \mathbf{E}[P(x_i > 0, x_j > 0 | \hat{x}_{i,j})]$, where $\hat{x}_{i,j}$ is the vector of x without the i th and j th elements, and the expectation is over the distribution of $\hat{x}_{i,j}$ conditioned on the condition $\forall k \notin i, j \sigma_k x_k > 0$.

The conditional distribution $P(x_i, x_j | \hat{x}_{i,j})$ is also a Gaussian with a, generally non-zero mean μ , and a covariance matrix given by

$$\bar{\Sigma} = \begin{pmatrix} \Sigma_{ii} & \Sigma_{ji} \\ \Sigma_{ij} & \Sigma_{jj} \end{pmatrix} - \hat{\Sigma},$$

where $\hat{\Sigma}$ is independent of Σ_{ij} . This means that increasing the Σ_{ji} (keeping Σ_{ii} and Σ_{jj} fixed) will increase the correlation in $\bar{\Sigma}$. Therefore, by Theorem C.4.3, $P(x_i > 0, x_j > 0 | \hat{x}_{i,j})$ increases, and thus P_{00}^{ij} increases. □

Corollary C.4.5. *An immediate consequence of Theorem C.4.4 is that $P(\forall i, x_i > 0) = P(x_i > 0, x_j > 0 | \forall k \notin \{i, j\} x_k > 0) P(\forall k \notin \{i, j\} x_k > 0)$ increases when $\Sigma_{ij} / \sqrt{\Sigma_{ii}\Sigma_{jj}}$ increases, as $P(\forall k \notin \{i, j\} x_k > 0)$ stays constant.*

Lemma C.4.6. *In the same setting as Theorem C.4.4, for covariance matrices Σ and Σ' of full rank, the following holds: for every i, j , $\Sigma_{ij} = \Sigma'_{ij}$ implies $P_{K,00}^{ij} = P_{K',00}^{ij}$ and $\Sigma_{ij}/\sqrt{\Sigma_{ii}\Sigma_{jj}} > \Sigma'_{ij}/\sqrt{\Sigma'_{ii}\Sigma'_{jj}}$ implies $P_{K,00}^{ij} > P_{K',00}^{ij}$, where P_K is the probability measure corresponding to covariance K .*

Proof. The set \mathbb{S} of all symmetric positive definite matrices is an open convex subset of the set of all matrices. Therefore, it is path-connected. We can traverse the path between Σ and Σ' through a sequence of points such that the distance between point x_i and x_{i+1} is smaller than the radius of a ball centered around x_i and contained in the set \mathbb{S} . Within this ball, one can move between x_i and x_{i+1} in coordinate steps that only change one element of the matrix. Therefore we can apply Theorem C.4.4 to each step of this path, which implies the theorem. \square

Corollary C.4.7. *Consider any set of m points $x^{(i)} \in \mathbb{R}^n$ with elements $x^{(i)}_j$. Let X be the $m \times n$ data matrix with $X_{ij} = x^{(i)}_j$. For a weight vector $w \in \mathbb{R}^n$ let $y = Xw$ be the vector of real-valued outputs of an perceptron, with weights sampled from an isotropic Gaussian $\mathcal{N}(0, I_n)$. If the correlation between any two inputs increases, then $P(T = 0) = P([\sum_s \mathbf{1}(s)] = 0)$ increases*

Proof. The vector $y = \sum_i X_i \cdot w_i$ is a sum of Gaussian vectors (with covariances of rank 1), and therefore is itself Gaussian, with a covariance given by $\Sigma = XX^T$. If the correlation product between any two inputs increases, then applying Theorem C.4.6 and Theorem C.4.5 at each step, the theorem follows. \square

Lemma C.4.8. *If the uncentered moments of the distribution $P(T = t)$ increase, except for its mean (which is 2^{n-1}), then $\langle H(t) \rangle$ increases.*

Proof. We consider the definition of the entropy, H , of a string (Theorem 3.2.3). We define the first and second terms by $h_1 = (t/t_{max}) \ln(t/t_{max})$ and $h_2 = (1 - t/t_{max}) \ln(1 - t/t_{max})$. We taylor expand h_2 about $t = 0$:

$$(1 - t/t_{max}) \ln(1 - t/t_{max}) = (1 - t/t_{max}) \sum_k \frac{1}{k} \left(-\frac{t}{t_{max}} \right)^k = \sum_k \left(\frac{1}{k} + \frac{1}{k-1} \right) \left(\frac{t}{t_{max}} \right)^k$$

By symmetry, we see that $h_1(t) = h_2(t_{max} - t)$, and we can thus Taylor expand h_1 around 0 t_{max} , to give:

$$\langle H(t) \rangle = \left\langle \sum_k a_k (t^k + (t_{max} - t)^k) \right\rangle = \left\langle \sum_k 2a_{2k} t^{2k} \right\rangle = \sum_k 2a_{2k} \langle t^{2k} \rangle$$

Because every $a_{2k} = \frac{1}{2k(2k-1)}$ is positive, we see that increasing every even moment increases the average entropy, $\langle H(t) \rangle$. \square

C.5 Proof of Theorem 4.4.1

Proof. Consider a concept c with generalization error $\epsilon(c)$. The probability that it has zero training error for a sample of m instances is $P[c \in C(S)] = (1 - \epsilon(c))^{-m}$. This is smaller or equal to some $\delta > 0$ if

$$-\ln(1 - \epsilon(c)) \geq \frac{\ln \frac{1}{\delta}}{m}$$

This can be written as follows.

$$\forall c \forall \delta > 0 \forall^\delta S \left[c \in C(S) \text{ implies } -\ln(1 - \epsilon(c)) < \frac{\ln \frac{1}{\delta}}{m} \right]$$

By the quantifier reversal lemma McAllester [1998],

$$\forall^\delta S \forall \alpha > 0 \forall^\alpha c \left[c \in C(S) \text{ implies } -\ln(1 - \epsilon(c)) < \frac{\ln \frac{1}{\alpha \beta \delta}}{(1 - \beta)m} \right]$$

Let $B(S) \subseteq \mathcal{H}$ be the set of concepts violating the formula, then $P(B(S)) \leq \alpha$. Now, the conditional probability of a concept in $C(S)$ violating the formula is $\frac{P(B(S) \cap C(S))}{P(C(S))} \leq \frac{P(B(S))}{P(C(S))} \leq \frac{\alpha}{P(C(S))}$. This probability is therefore smaller than γ if $\alpha = \gamma P(C(S))$. We thus get

$$\forall^\delta S \forall \alpha > 0 \forall^\gamma c \in C(S) \left[-\ln(1 - \epsilon(c)) < \frac{\ln \frac{1}{P(C(S))} + \ln \frac{1}{\gamma \beta \delta}}{(1 - \beta)m} \right]$$

We get the result, choosing $\beta = \frac{1}{m}$

\square

C.6 Proof of Theorem 4.5.1

Proof. We begin by decomposing $P(S_{m+1})$ as follows

$$P(S_{m+1}) = P(S_m)(1 - P_e(m))$$

Denote $P_m = P(S_m)$. Take the natural logarithm of this and average over S_m to obtain

$$-\langle \log(1 - P_e(m)) \rangle = \langle \log P_m \rangle - \langle \log P_{m+1} \rangle$$

Using $-\log(1 - P_e(m)) \geq P_e(m)$, we obtain $\langle \epsilon(m) \rangle \leq \langle \log P_m \rangle - \langle \log P_{m+1} \rangle$.

Now, using Markov's inequality, we know that $\mathbf{P}[P_e(m) > 0.6] < \frac{\langle \epsilon(m) \rangle}{0.6}$. By our assumption, $P_e(m) < E$. By letting $E' = -\log(1 - E)$, and using the fact that $-\log 1 - x \leq x + x^2$ for $x < 0.6$, we can write

$$-\langle \log(1 - P_e(m)) \rangle \leq \langle \epsilon(m) \rangle + \langle P_e(m)^2 \rangle + \frac{\langle \epsilon(m) \rangle}{0.6} E'$$

As $P_e(m) \leq 1$, $\langle P_e(m)^2 \rangle \leq \langle P_e(m) \rangle = \langle \epsilon(m) \rangle$, and we obtain the first statement of the theorem.

To obtain the second, we use Chebysev's inequality, which says $\mathbf{P}[P_e(m) > 0.6] < \frac{\text{Var}(P_e(m))}{(0.6 - \langle \epsilon(m) \rangle)^2}$. Following the same decomposition as above (into cases where $P_e(m) \leq 0.6$ and $P_e(m) > 0.6$), we have

$$-\langle \log(1 - P_e(m)) \rangle \leq \langle \epsilon(m) \rangle + \langle P_e(m)^2 \rangle + \frac{\text{Var}(P_e(m))}{(0.6 - \langle \epsilon(m) \rangle)^2} E'$$

Now, $\langle P_e(m)^2 \rangle = \text{Var}(P_e(m)) + \langle P_e(m) \rangle^2$ and $\langle P_e(m) \rangle^2 = o(\langle P_e(m) \rangle)$. Also, $\langle \epsilon(m) \rangle = o(1)$. Therefore, as long as $\text{Var}(P_e(m)) = o(\langle \epsilon(m) \rangle)$, we have

$$-\langle \log(1 - P_e(m)) \rangle \leq \langle \epsilon(m) \rangle + o(\langle \epsilon(m) \rangle),$$

and the second part of the theorem follows. □

C.7 Derivation of KL divergence inequality

Consider a parametrized family of functions $\mathcal{F} = \{f_\theta\}_{\theta \in \Theta}$ parametrized by parameters $\theta \in \Theta$ (for example the set of functions expressible by a DNN). We can define the parameter-function map as in Valle-Pérez et al. [2018] \mathcal{M} as:

$$\mathcal{M} : \Theta \rightarrow \mathcal{F}$$

$$\theta \mapsto f_\theta.$$

We can then define the pre-image $\mathcal{M}^{-1}(f)$ which is the set of all θ which produce f under \mathcal{M} . Furthermore, for any distribution P on Θ , we define $\bar{P}(f) := P(\mathcal{M}^{-1}(f)) = \sum_{\theta \in \mathcal{M}^{-1}(f)} P(\theta)$

Let Q , and P be two distributions on Θ , then the KL divergence is

$$\begin{aligned} KL(Q||P) &= - \sum_{\theta \in \Theta} Q(\theta) \log \left(\frac{P(\theta)}{Q(\theta)} \right) \\ &= - \sum_{f \in \mathcal{F}} \bar{Q}(f) \sum_{\theta \in \mathcal{M}^{-1}(f)} \frac{Q(\theta)}{\bar{Q}(f)} \log \left(\frac{P(\theta)}{Q(\theta)} \right) \\ &\geq - \sum_{f \in \mathcal{F}} \bar{Q}(f) \log \left(\sum_{\theta \in \mathcal{M}^{-1}(f)} \frac{Q(\theta)}{\bar{Q}(f)} \frac{P(\theta)}{Q(\theta)} \right) \\ &= - \sum_{f \in \mathcal{F}} \bar{Q}(f) \log \left(\frac{\bar{P}(f)}{\bar{Q}(f)} \right) \\ &= KL(\bar{Q}||\bar{P}) \end{aligned}$$

where we the third line follows from Jensen's inequality. We thus see that the KL divergence between distributions in parameter space is no less than the KL divergence between the induced distributions in parameter space.

Appendix D

Further results for the simplicity bias in the simple Boolean system

In this appendix, we give further results related to the PF map of the system Boolean system studied in Chapter 2.

D.1 Other complexity measures

One of the key steps to practical application of the simplicity bias framework of Dingle et al. in Dingle et al. [2018a] is the identification of a suitable complexity measure $\tilde{K}(x)$ which mimics aspects of the (uncomputable) Kolmogorov complexity $K(x)$ for the problem being studied. It was shown for the maps in Dingle et al. [2018a] that several different complexity measures all generated the same qualitative simplicity bias behaviour:

$$P(x) \leq 2^{-(a\tilde{K}(x)+b)} \quad (\text{D.1})$$

but with different values of a and b depending on the complexity measure and of course depending on the map, but independent of output x . Showing that the same qualitative results obtain for different complexity measures is sign of robustness for simplicity bias.

In this section we performed experiments looking at simplicity bias when using different complexity measures which can be used for Boolean functions, which we define in Section D.1.1.

We find they give similar results, but some complexity measures correlate better with the probability $P(f)$ than others. In Section D.1.3, we show that probability correlates with the size of the smallest Boolean expression expressing that function, as well as with two complexity measures related to the sensitivity of the output to changes in the input. In Figure D.1, we also compare the different measures to each other. We can see that they all correlate, but also show differences in their ability to recognize regularity. Which complexity measures best capture real-world regularity remains an open question.

In the next section, we also perform several experiments, for different complexity measures, showing the effect of the complexity of the target function on learning.

D.1.1 Complexity measures

Lempel-Ziv complexity (LZ complexity for short). The Boolean functions studied in the previous section can be written as binary strings, which makes it possible to use measures of complexity based on finding regularities in binary strings. One of the best is Lempel-Ziv complexity, based on the Lempel-Ziv compression algorithm. It has many nice properties, like asymptotic optimality, and being asymptotically equal to the Kolmogorov complexity for an ergodic source. We use the variation of Lempel-Ziv complexity from Dingle et al. [2018a] which is based on the 1976 Lempel Ziv algorithm (Lempel and Ziv [1976]):

$$K_{LZ}(x) = \begin{cases} \log_2(n), & x = 0^n \text{ or } 1^n \\ \log_2(n)[N_w(x_1 \dots x_n) + N_w(x_n \dots x_1)]/2, & \text{otherwise} \end{cases} \quad (\text{D.2})$$

where n is the length of the binary string, and $N_w(x_1 \dots x_n)$ is the number of words in the Lempel-Ziv "dictionary" when it compresses output x . The symmetrization makes the measure more fine-grained, and the $\log_2(n)$ factor as well as the value for the simplest strings ensures that they scale as expected for Kolmogorov complexity. This complexity measure is the primary one used in our experiments in this chapter.

We note that the binary string representation depends on the order in which inputs are listed to construct it, which is not a feature of the function itself. This may affect the LZ complexity, although for low-complexity input orderings (we use numerical ordering of the

binary inputs), it has a negligible effect, so that $K(x)$ will be very close to the Kolmogorov complexity of the function.

Entropy. A fundamental, though weak, measure of complexity is the entropy. For a given binary string this is defined as $S = -\frac{n_0}{N} \log_2 \frac{n_0}{N} - \frac{n_1}{N} \log_2 \frac{n_1}{N}$, where n_0 is the number of zeros in the string, and n_1 is the number of ones, and $N = n_0 + n_1$. This measure is close to 1 when the number of ones and zeros is similar, and is close to 0 when the string is mostly ones, or mostly zeros. Entropy and $K_{LZ}(x)$ are compared in Fig. D.1, and in more detail in supplementary note 7 (and supplementary information figure 1) of reference Dingle et al. [2018a]. They correlate, in the sense that low entropy $S(x)$ means low $K_{LZ}(x)$, but it is also possible to have Large entropy but low $K_{LZ}(x)$, for example for a string such as 10101010....

Boolean expression complexity. Boolean functions can be compressed by finding simpler ways to represent them. We used the standard SciPy implementation of the Quine-McCluskey algorithm to minimize the Boolean function into a small sum of products form, and then defined the number of operations in the resulting Boolean expression as a *Boolean complexity* measure.

Generalization complexity. L. Franco et al. have introduced a complexity measure for Boolean functions, designed to capture how difficult the function is to learn and generalize (Franco and Anthony [2004]), which was used to empirically find that simple functions generalize better in a neural network (Franco [2006]). The measure consists of a sum of terms, each measuring the average over all inputs fraction of neighbours which change the output. The first term considers neighbours at Hamming distance of 1, the second at Hamming distance of 2 and so on. The first term is also known (up to a normalization constant) as average sensitivity (Friedgut [1998]). The terms in the series have also been called “generalized robustness” in the evolutionary theory literature (Greenbury et al. [2016]). Here we use the first two terms, so the measure is:

$$C(f) = C_1(f) + C_2(f),$$

$$C_1(f) = \frac{1}{2^n n} \sum_{x \in X} \sum_{y \in \text{Nei}_1(x)} |f(x) - f(y)|,$$

$$C_2(f) = \frac{2}{2^n n(n-1)} \sum_{x \in X} \sum_{y \in \text{Nei}_2(x)} |f(x) - f(y)|,$$

where $\text{Nei}_i(x)$ is all neighbours of x at Hamming distance i .

Critical sample ratio. A measure of the complexity of a function was introduced in Krueger et al. [2017a] to explore the dependence of generalization with complexity. In general, it is defined with respect to a sample of inputs as the fraction of those samples which are *critical samples*, defined to be an input such that there is another input within a ball of radius r , producing a different output (for discrete outputs). For Boolean functions, we extend this definition and define it as the fraction of all inputs, that have another input at Hamming distance 1, producing a different output.

D.1.2 Correlation between complexities

In Fig. D.1, we compare the different complexity measures against one another. We also plot the frequency of each complexity; generally more functions are found with higher complexity.

D.1.3 Probability-complexity plots

In Fig. D.2 we show how the probability versus complexity plots look for other complexity measures. The behaviour is similar to that seen for the LZ complexity measure in Figure 2.1b.

D.2 Effect of number of layers on simplicity bias

In Figure D.3 we show the effect of the number of layers on the bias (for feedforward neural networks with 40 neurons per layer). The left figures show the probability of individual functions versus the complexity. The right figure shows the histogram of complexities, weighted by the

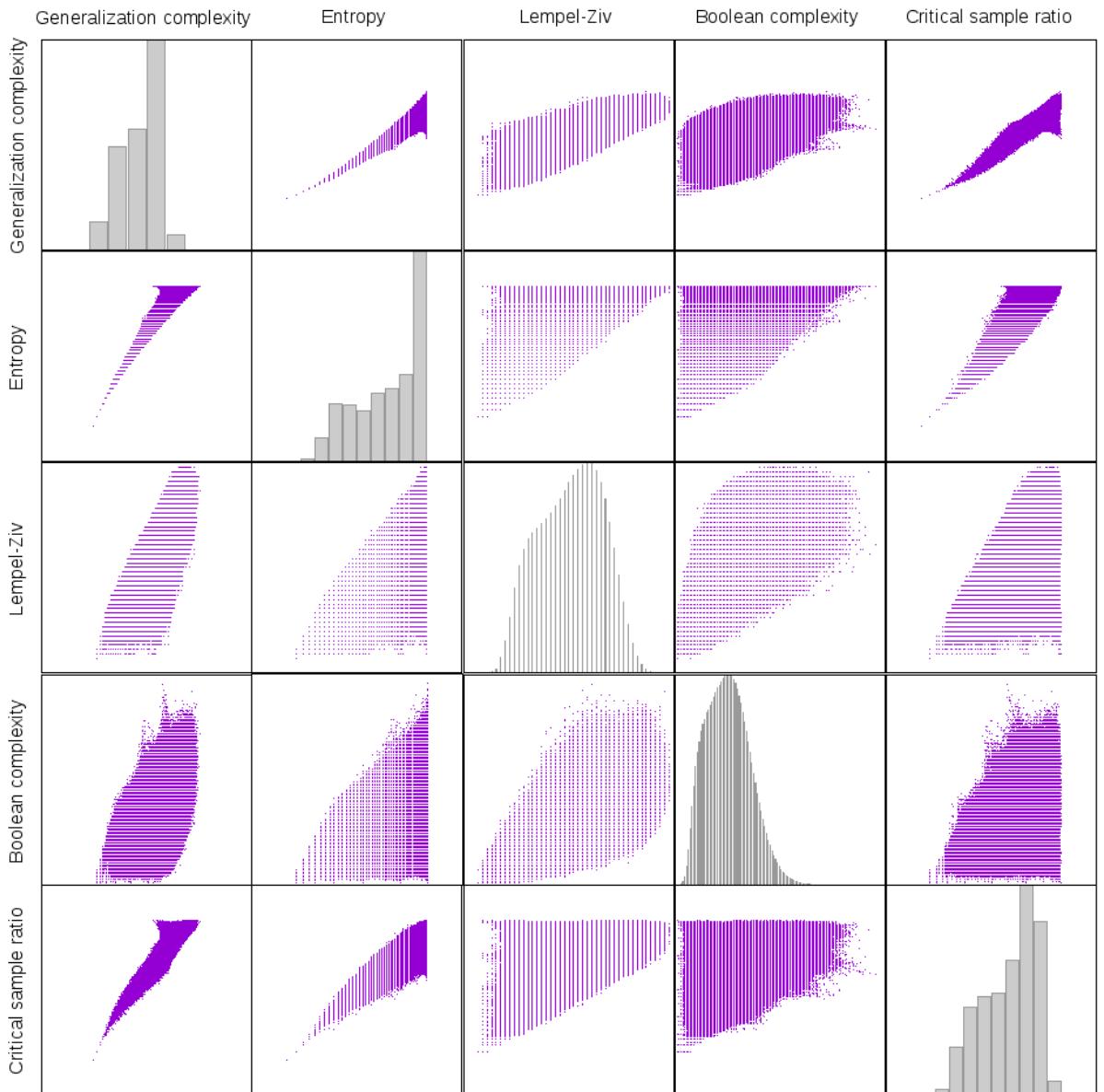
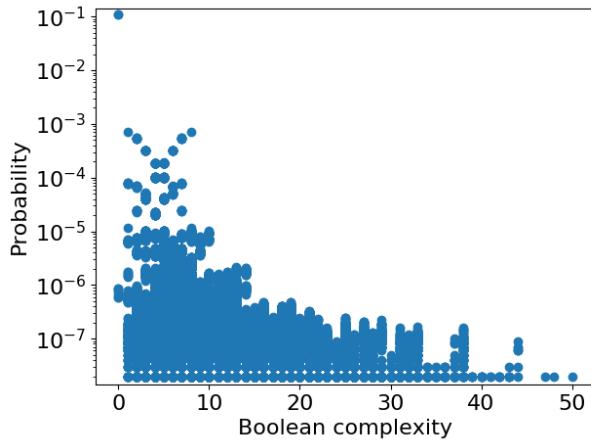
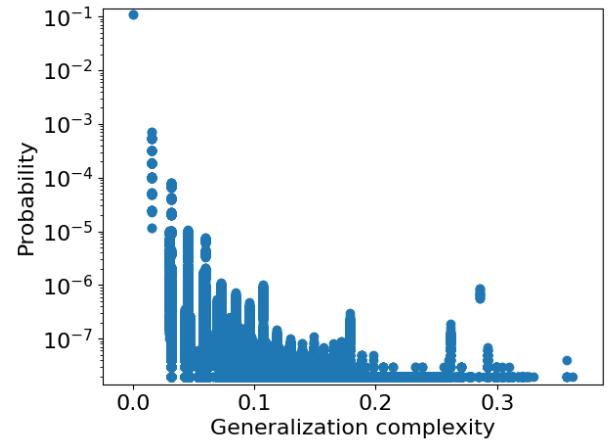


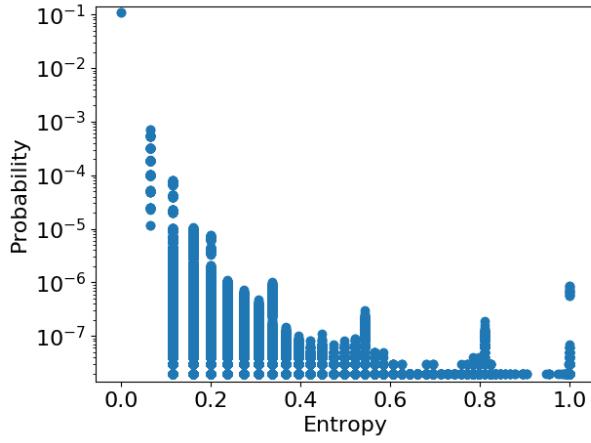
Figure D.1: Scatter matrix showing the correlation between the different complexity measures used in this paper. On the diagonal, a histogram (in grey) of frequency versus complexity is depicted. The functions are from the sample of 10^8 parameters for the $(7, 40, 40, 1)$ network.



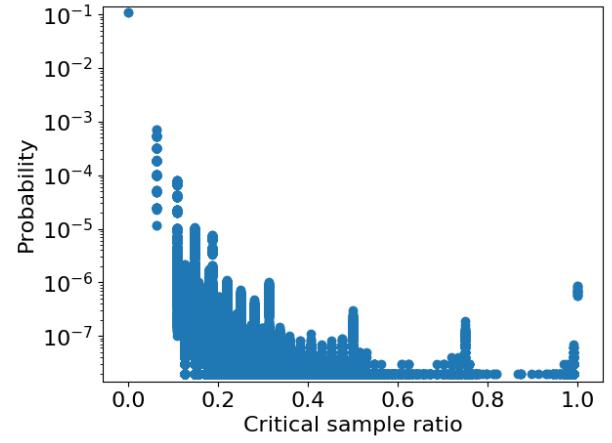
(a) Probability versus Boolean complexity



(b) Probability versus generalization complexity



(c) Probability versus entropy



(d) Probability versus critical sample ratio

Figure D.2: Probability versus different measures of complexity (see Figure 2.1b for Lempel-Ziv), estimated from a sample of 10^8 parameters, for a network of shape $(7, 40, 40, 1)$. Points with a frequency of 10^{-8} are removed for clarity because these suffer from finite-size effects (see Section A.1.1). The measures of complexity are described in Section D.1.1.

probability by which the function appeared in the sample of parameters. The histograms therefore show the distribution over complexities when randomly sampling parameters¹ We can see that between the 0 layer perceptron and the 2 layer network there is an increased number of higher complexity functions. This is most likely because of the increasing expressivity of the network. For 2 layers and above, the expressivity does not significantly change, and instead, we observe a shift of the distribution towards lower complexity.

¹using a Gaussian with $1/\sqrt{n}$ variance in this case, n being number of inputs to neuron

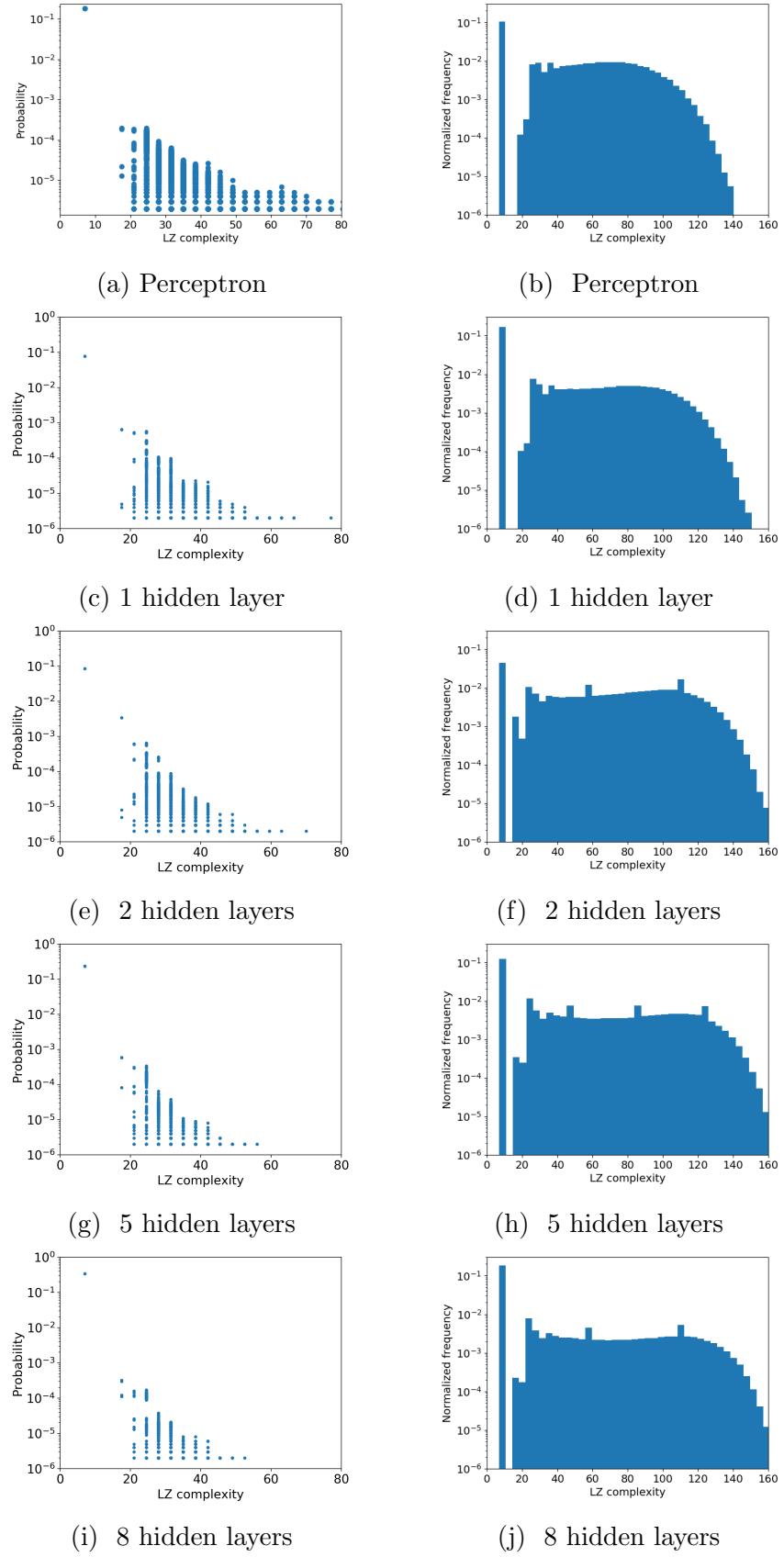


Figure D.3: Probability versus LZ complexity for networks with different number of layers. Samples are of size 10^6 (a) & (b) A perceptron with 7 input neurons (complexity is capped at 80 in (a) to aid comparison with the other figures). (c) to (j) are networks with 40 hidden neurons per layer

D.3 Effect of target function complexity on learning for different complexities

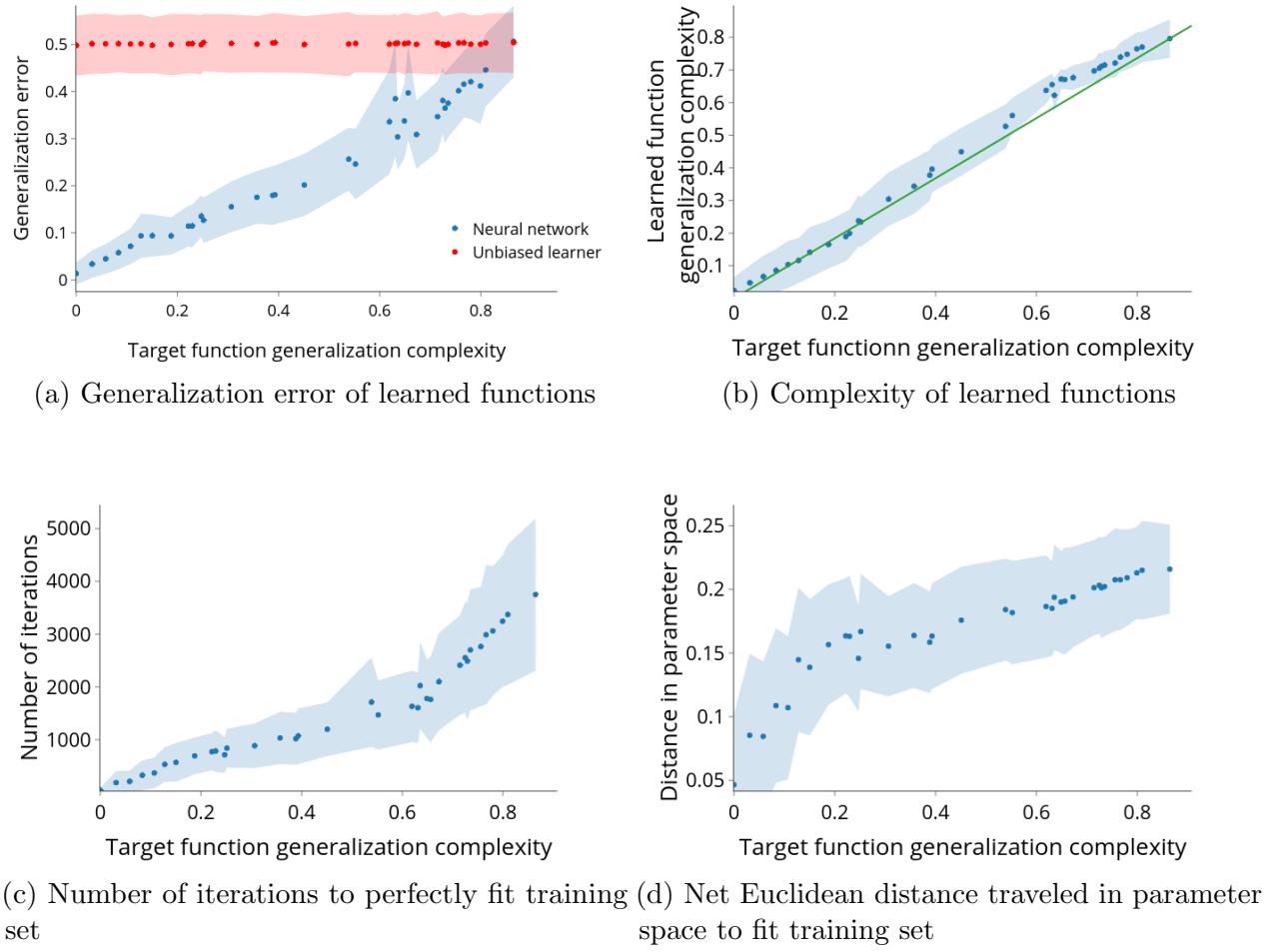


Figure D.4: Different learning metrics versus the generalization complexity of the target function, when learning with a network of shape $(7, 40, 40, 1)$. Dots represent the means, while the shaded envelope corresponds to piecewise linear interpolation of the standard deviation, over 500 random initializations and training sets.

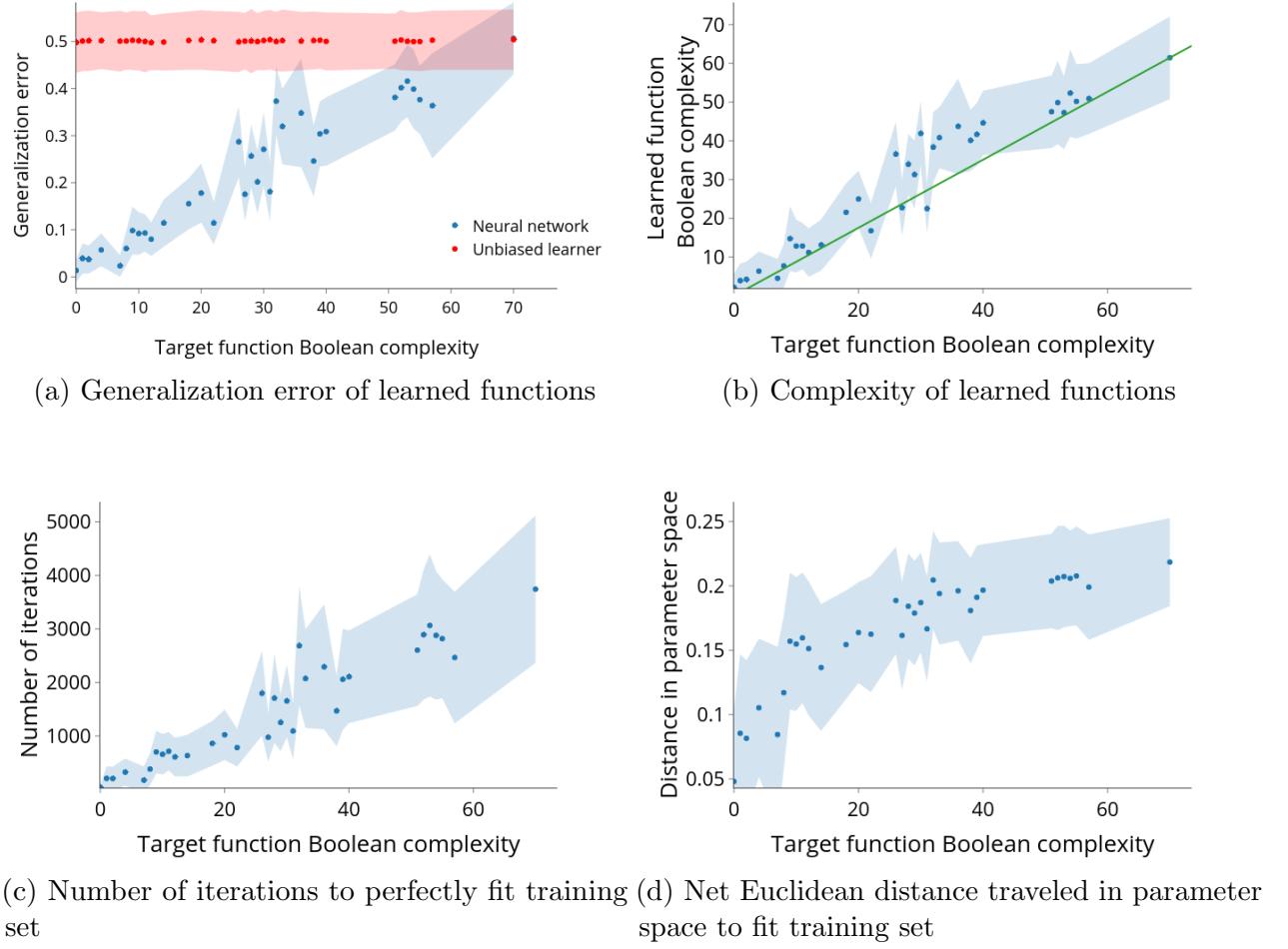


Figure D.5: Different learning metrics versus the Boolean complexity of the target function, when learning with a network of shape $(7, 40, 40, 1)$. Dots represent the means, while the shaded envelope corresponds to piecewise linear interpolation of the standard deviation, over 500 random initializations and training sets.

D.4 Probability-complexity plots for other parameter distributions

In Fig. D.7 we show probability versus LZ complexity plots for other choices of parameter distributions.

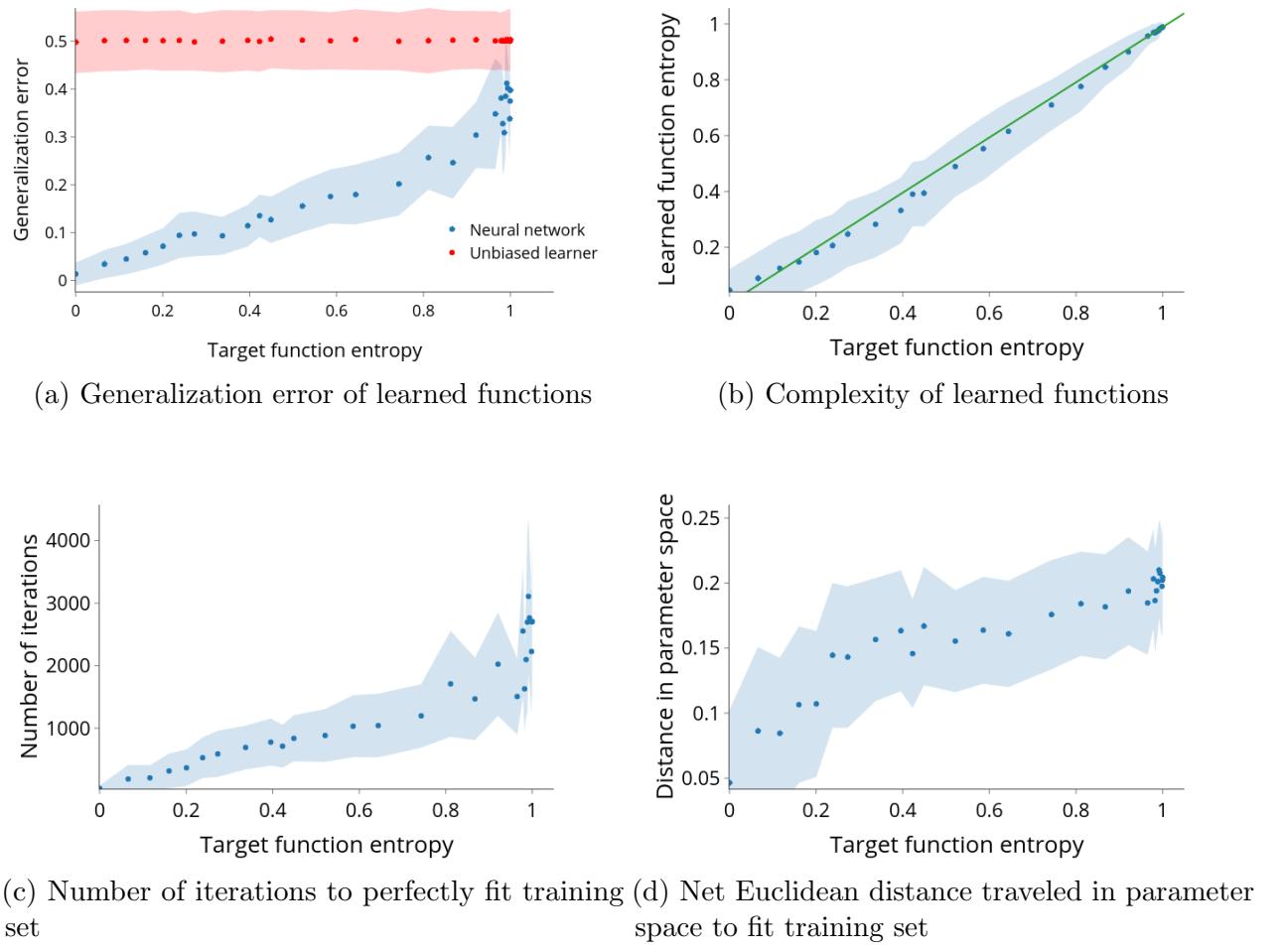


Figure D.6: Different learning metrics versus the entropy of the target function, when learning with a network of shape $(7, 40, 40, 1)$. Dots represent the means, while the shaded envelope corresponds to piecewise linear interpolation of the standard deviation, over 500 random initializations and training sets.

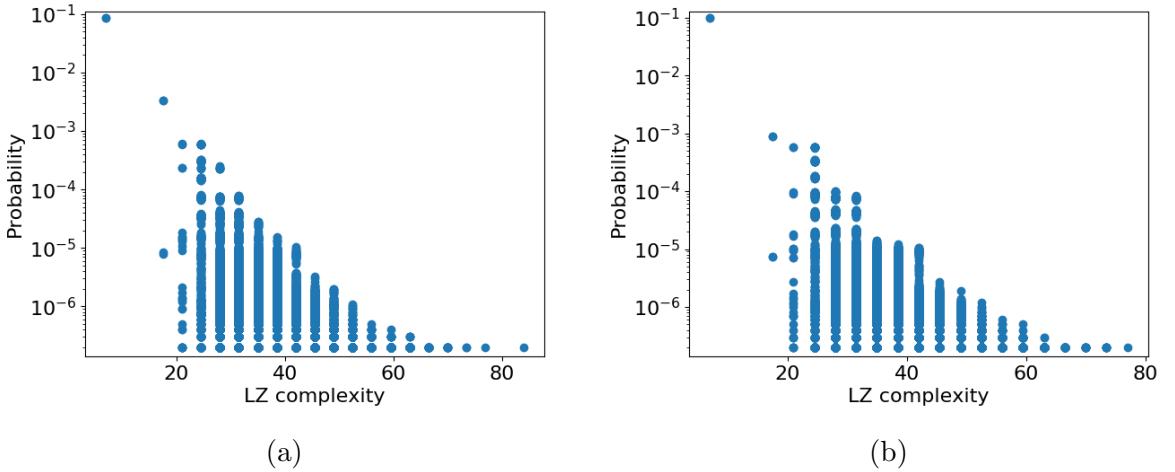


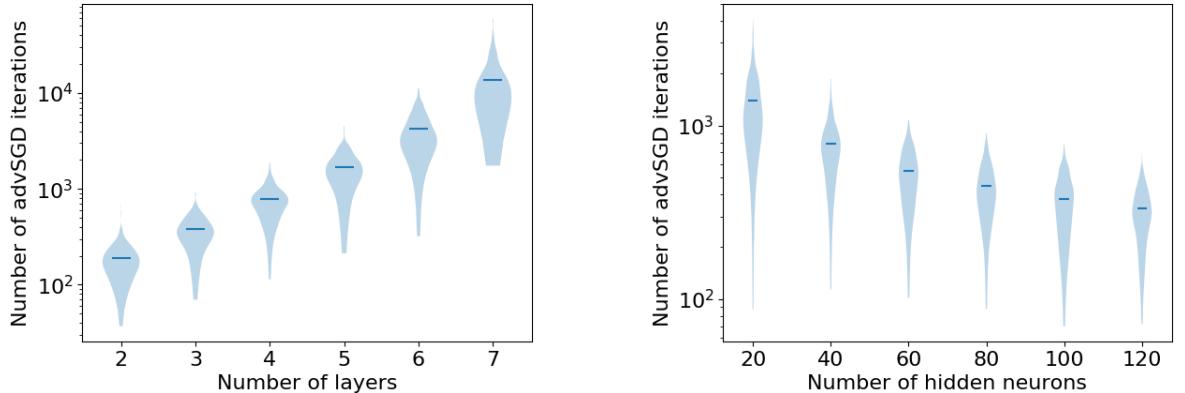
Figure D.7: Probability versus LZ complexity for network of shape $(7, 40, 40, 1)$ and varying sampling distributions. Samples are of size 10^7 . (a) Weights are sampled from a Gaussian with variance $1/\sqrt{n}$ where n is the input dimension of each layer. (b) Weights are sampled from a Gaussian with variance 2.5

D.5 Number of iterations to learn a function vs depth and width

Although our main focus is on generalization, in this section we show some of the effects that changing the PF map can have on optimization. In Figure D.8, we see that increasing the number of hidden neurons per layer tends to decrease the training time, while increasing the number of layers tends to increase the training time. For the case of deep linear networks, Advani and Saxe [2017] have derived some theoretical results which show that in that case more layers result in longer training time. Schoenholz et al. [2017] have connected the longer training time for deeper networks to their information propagation, and a presence of “chaotic regime” (see Poole et al. [2016]). The intuition is that if $P(f)$ becomes less biased, it also becomes harder to find functions that fit the data, as all functions are getting a similar “share” of parameter space, and most functions do not fit the data.

Figure D.8b shows that wider networks learn faster, this has also been observed in the experiments on scaling laws in recent work [Kaplan et al., 2020, Henighan et al., 2020]. While we currently don’t have a full explanation for this effect, it may be related to some “blessing of dimensionality”, whereby minima become connected with higher overparametrization [Garipov

et al., 2018, Draxler et al., 2018], which probably helps the optimizer find better minima quicker.



(a) Number of iterations of advSGD to perfectly fit a training set for a network with 40 hidden neurons per layer, and varying number of layers

(b) Number of iterations of advSGD to perfectly fit a training set for a network with 4 layers, and varying number of hidden neurons per layer

Figure D.8: Number of iterations of advSGD to perfectly fit a training set from a fixed function versus either number of layers or hidden neurons per layer, for a network with 7 inputs and 1 output. Both are for a fixed target function with LZ complexity 31.5, and with a training set of size 64. The violins show the data for 200 random initializations and training sets, the line showing the mean.

D.6 Bias and the curse of dimensionality

We have argued in Chapter 2 that the main reason deep neural networks are able to generalize is because their implicit prior over functions is heavily biased. We base this claim on the argument (for example justified by PAC-Bayes in Section 4.4) that large bias towards the right kind of functions implies generalization. The contrapositive of this claim is that bad generalization implies small bias, or bias towards the wrong kind of functions. Here we describe some examples of this, connecting them to the curse of dimensionality.

Complex machine learning tasks require models which are expressive enough to be able to learn the target function. For this reason, before deep learning, the main approach to complex tasks was to use non-parametric models which are infinitely expressible. These include Gaussian processes and other kernel methods. However, unlike deep learning, these models were not successful when applied to tasks where the dimensionality of the input space was very

large. We therefore expect that these models show little bias, as they generalize poorly.

Many of these models use kernels which encode some notion of local continuity. For example, the Gaussian kernel ensures that points within a ball of radius λ are highly correlated. On the other hand, points separated by a distance greater than λ can be very different. Intuitively, we can divide the space into regions of length scale λ . If the input domain we're considering has $O(1)$ volume, and has dimensionality d (is a subset of \mathbb{R}^d), then the volume of each of these regions is of order λ^d , and the number of these regions is of order $1/\lambda^d$. In the case of binary classification, we can estimate the effective number of functions which the kernel “prefers” by constraining the function to take label 0 or 1 within each region, but with no further constraint. The number of such functions is 2^{a^d} , where we let $a := 1/\lambda$. Each of these functions is equally likely, and together they take the bulk of the total probability, so that they have probability close to 2^{-a^d} , which decreases very quickly with dimension.

Kernels like the Gaussian kernel are biased towards functions which are locally continuous. However, for high dimension d , they are not biased *enough*. In particular, as the probability of the most likely functions grows doubly exponentially with d , we expect PAC-Bayes-like bounds (Section 4.4) to grow exponentially with d , quickly becoming vacuous. This argument is essentially a way of understanding the curse of dimensionality from the perspective of priors over functions.

Appendix E

Further results for the perceptron

In this appendix, we give further results related to the PF map of the perceptron, studied in Chapter 3.

E.0.1 Zipf's law in a perceptron with $b = 0$

In Chapter 2, we saw empirical evidence that the rank plot for a simple DNN exhibited a Zipf like power law scaling for larger ranks. Zipf's law occurs in many branches of science (and probably for many reasons). In this section we check whether this scaling also occurs for the Perceptron.

In Figure E.1, we compare a rank plot of the probability $P(f)$ for individual functions for the simple perceptron with $b = 0$, the perceptron, and for a one layer FCN. While all architectures have $n = 7$, the perceptrons can of course express far fewer functions. Nevertheless, both the perceptrons and the more complex FCN show similar phenomenology, with a Zipf law like tail at larger ranks (i.e. a power law).

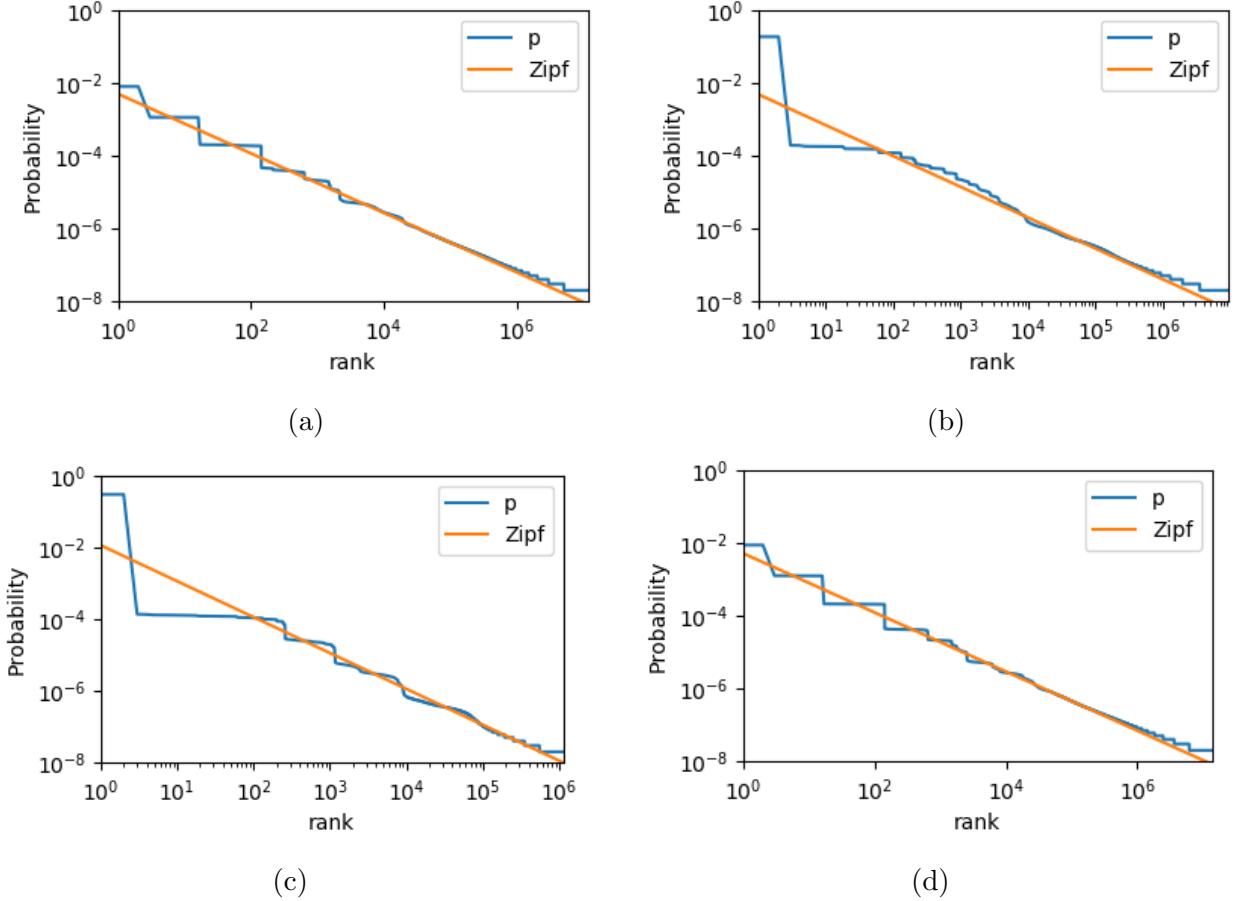


Figure E.1: Probability vs rank for functions (ranked by probability) from samples of size 10^8 , with input size $n = 7$, and every weight and bias term sampled from $\mathcal{N}(0, 1)$ unless otherwise specified, over initialisations of: (a) a perceptron with $b = 0$; (b) a perceptron; (c) a one-hidden layer neural network (with 64 neurons in the hidden layer); (d) a perceptron with $b = 0$ and weights sampled from identical centered uniform distributions (note how similar (a) is to (d)!). We cut off frequencies less than 2 to eliminate finite size effects. In (a) and (b) lines were fitted using least-squares regression; for (c) the line corresponding to the ansatz in Equation (E.1) is plotted instead.

While the original formulations for Zipf's law only allows for a powerlaw with exponent 1, in practice the terminology of Zipf's law is used for other powers, such that $p = b \times \text{rank}^{-a}$ for some positive a, b . If we assume that this scaling persists, then we can relate the total number of functions a perceptron can express, to the constant b , because the total probability must integrate to 1.

For the simplest case with $a = 1$, this leads to an equation for the probability as function of

rank given by

$$P(r) = \frac{1}{\ln(N_O)r}, \quad (\text{E.1})$$

where N_O is the total number of functions expressible.

The FCN appears to show such simple scaling. And as the FCN of width 64 is fully expressive (see Theorem 3.4.1), there are $N_O = 2^7 \approx 3 \times 10^{38}$ possible Boolean functions. We plot the Zipf law prediction of Equation (E.1) next to the empirically estimated probabilities in Figure E.1c. We observe that the curve is described well at higher values of the rank Zipf's law. Note that the mean probability uniformly sampled over functions for this FCN is $\langle P(f) \rangle = 1/N_O \approx 3 \times 10^{-39}$ so that we only measure a tiny fraction of the functions with extremely high probabilities, compared to the mean. Also, most functions have probabilities less than the mean, and only order 2^{-n} have probability larger than the mean. A least-squares linear fit on the log-log graph was consistent within experimental error for the ansatz.

For the perceptron with a bias term Figure E.1b, we observe that the gradient differs substantially from -1 , and a linear fit gives $\log_{10}(p) = -0.85 \log_{10}(\text{rank}) - 2.32$. Using the same arguments for calculating N_O as made in Equation (E.1), we obtain a prediction of $N_0 = 7.63 \times 10^9$, which is 91% of the known value¹. A linear fit for the perceptron with no threshold bias term gives $\log_{10}(p) = -0.81 \log_{10}(\text{rank}) - 2.31$, leading to a prediction of $N_0 = 3.97 \times 10^8$, which is, as expected, significantly lower than a perceptron with no threshold bias term. We expect there to be some discrepancy between the pure Zipf law prediction, and the true N_O , because the probability seems to deviate from the Zipf-like behaviour at the highest rank, which we observe for $n = 5$ in Figure E.2a, as in this case the number of functions is small enough that it becomes feasible to sample all of them.

It is also worth mentioning that a rank-probability plot for a perceptron with weights sampled from a uniform distribution (Figure E.1d) is almost indistinguishable from the Gaussian case (Figure E.1b), which is interesting, because when plotted against LZ complexity, as in Figure 3.1 of Chapter 3, there is a small but discernible difference between the two types of initialisation.

Finally, in Figure E.2a we compare the rank plot for centered and uncentered data for a smaller $n = 5$, $\sigma_b = 0$ perceptron where we can find all functions. Note that for the centered

¹<https://oeis.org/A000609/list>

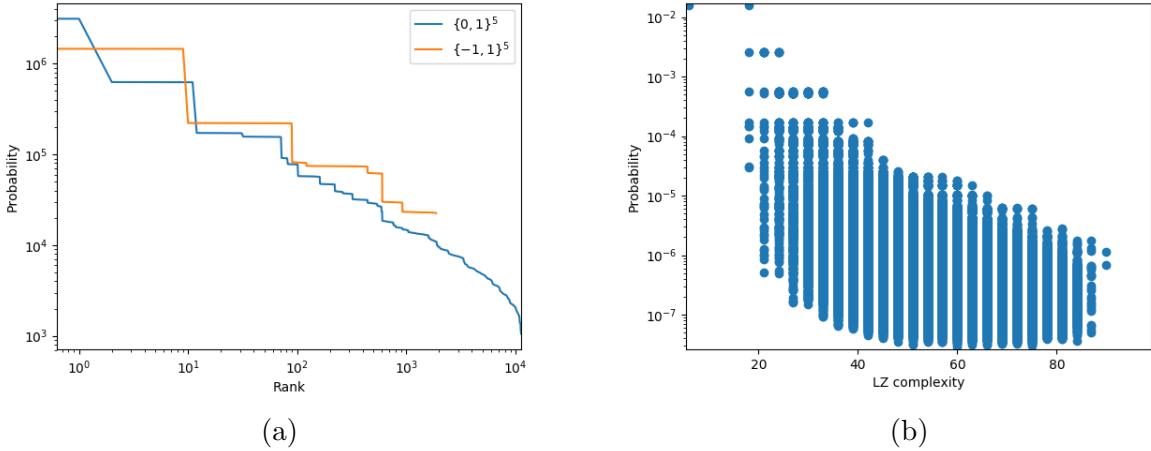


Figure E.2: (a) Probability of functions versus their rank (ranked by probability) for a perceptron with $n = 5$, and weights sampled i.i.d. from a Gaussian and no threshold bias term, acting on either centered $\{-1, 1\}^5$ or uncentered $\{0, 1\}^5$ data. (b) Probability of functions versus their LZ complexity for a perceptron with $n = 6$, and weights sampled i.i.d. from a Gaussian and no threshold bias term, acting on the centered Boolean hypercube $\{-1, 1\}^5$.

data, only functions with $t = 16$ can be expressed, which is somewhat peculiar, and of course means significantly less functions. Nevertheless, this system still has clear bias within this one entropy class, and this bias correlates with the LZ complexity (Figure E.2b) as also observed for the perceptron with centered data.

E.1 Bias disappears in the chaotic regime

In Section 3.4, we analyzed fully connected networks with ReLU activations, and find a general bias towards low entropy for a wide range of parameters. In a stimulating study, Yang et al. ([Yang and Salman, 2019]) recently showed an example of a network using an erf activation function where bias disappeared with increasing number of layers.

Here we argue that the reason for this behaviour lies in the emergence of a chaotic regime, which does not occur for ReLU activations, but does for some other activation functions. In particular, the erf activation function is very similar to the tanh activation function used in an important series of recent papers that studied the propagation of correlations between hidden layer activations through neural networks, which they call “deep information propagation” ([Poole et al., 2016, Schoenholz et al., 2017, Lee et al., 2018]). They find that the activation

function can have a dramatic impact on the behaviour of correlations. In particular, these papers show that for FCNs with tanh activation, there are two distinct parameter regimes in the asymptotic limit of infinite depth: One in which inputs approach perfect correlation (the *ordered regime*), and one in which the inputs approach 0 correlation (the *chaotic regime*). FCNs with ReLU activation do not appear to exhibit this chaotic regime, although they nevertheless have different dynamic regimes.

The particular example in [Yang and Salman, 2019] was for $\sigma_w = 4.0$, and $\sigma_b = 0.0$, a choice of hyperparameters that, for sufficient depth, lies deep in the chaotic regime. In this regime, inputs with initial correlations > -1 and < 1 will become asymptotically uncorrelated for sufficient depth, while initial correlations with equal to ± 1 stay fixed, as we show below. The network is therefore equally likely to produce any odd function (on the $\{-1, 1\}^n$ Boolean hypercube).

To confirm our conjecture above, we performed experiments to calculate the bias of tanh networks in both the chaotic and ordered regime, which we show in Figure E.3. We obtain the expected results: in the chaotic regime, the bias gets weaker with number of layers, while in the ordered regime, just as was found for the ReLU activation in the Chapter 3, the bias remains, and the trivial functions gain more probability. These experiment illustrate for tanh activation that in either the chaotic or ordered regime, the *a-priori* bias of the network becomes asymptotically degenerate with depth, either by becoming unbiased, or too biased.

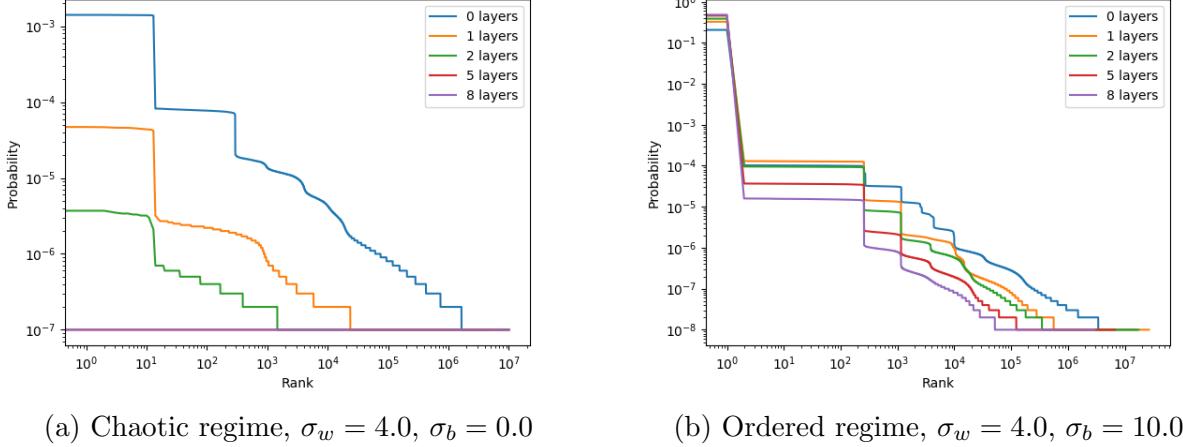


Figure E.3: Probability versus rank (ranked by probability) of different Boolean functions $\{-1, 1\}^7 \rightarrow \{-1, 1\}$ produced by a neural network with 1 hidden layer with tanh activation, and weights distributed by a Gaussian with different variance hyperparameters chosen to lie in the chaotic ($\sigma_b = 0.0$) and ordered ($\sigma_b = 10.0$) regimes.

We now explain that a simple extension of the analysis of [Poole et al., 2016] shows that, for the special case of $\sigma_b = 0.0$ and tanh activation function, initial correlations equal to ± 1 stay fixed. This happens because the RHS in Equation 5. describing the propagation of correlation in [Poole et al., 2016] (which we replicate in Equation (E.2) below) is odd on the correlation q_{12}^{l-1} (which represents the covariance between a pair of activations for inputs 1 and 2 at layer $l - 1$) when $\sigma_b = 0$. In addition to the fixed point they identified for the correlation being $+1$ there is therefore another unstable fixed point at -1 .

$$q_{12}^l = \mathcal{C}(c_{12}^{l-1}, q_{11}^{l-1}, q_{22}^{l-1} | \sigma_w, \sigma_b) \equiv \sigma_w^2 \int \mathcal{D}z_1 \mathcal{D}z_2 \phi(u_1) \phi(u_2) + \sigma_b^2 \quad (\text{E.2})$$

$$u_1 = \sqrt{q_{11}^{l-1}} z_1, u_2 = \sqrt{q_{22}^{l-1}} \left[c_{12}^{l-1} z_1 + \sqrt{1 - (c_{12}^{l-1})^2} z_2 \right], \quad (\text{E.3})$$

where $c_{12}^l = q_{12}^l (q_{11}^l q_{22}^l)^{-1}$, and z_1, z_2 are independent standard Gaussian variables, and the q_{11} and q_{22} are the variances of the activations for input 1 and input 2, respectively. This fixed point at -1 ensures that points which are parallel but opposite (like opposite corners in the $\{-1, 1\}^n$ hypercube) will stay perfectly anti-correlated. This agrees with the expectation that the erf/tanh network with $\sigma_b = 0$ can only produce odd functions, as the activations are odd functions. However, any other pair of points becomes uncorrelated, and this explains why

every (real valued) function, up to the oddness constraint, is equally likely. This also implies that every odd Boolean function is equally likely, as the region of function space satisfying the oddness constraint has the same shape within every octant corresponding to an odd Boolean function. This is because the region in one octant is related to that on another octant by simply changing signs of elements of the function vector (a reflection transformation).

As a future direction, it will be interesting to study the effect of these different dynamic regimes, for different activation functions, on simplicity bias.

E.2 Further results on the distribution within \mathbb{F}_t

In this section we will denote the output function of the perceptron evaluated on $\{0, 1\}^n$ by way of a bit string f , whose i 'th bit is given by

$$f_i = \mathbf{1}(\langle w, \text{bin}(i) \rangle) \quad (\text{E.4})$$

where $\text{bin}(i)$ takes an integer i and maps it to a point $x \in \{0, 1\}^n$ according to its binary representation (so $\text{bin}(5) = (1, 0, 1)$ and $\text{bin}(1) = (0, 0, 1)$ when $n = 3$).

E.2.1 Empirical results

We sample 10^8 initialisations of the perceptron, divide the list of functions into \mathbb{F}_t , and present probability-complexity plots for several values of t in Figure E.4. We use the Lempel-Ziv complexity [Lempel and Ziv, 1976, Dingle et al., 2018c] of the output bit string as the approximation to the Kolmogorov complexity of the function [Dingle et al., 2018c]. As in [Dingle et al., 2018c, Valle-Pérez et al., 2018], this complexity measure is denoted $K_{LZ}(f)$. To avoid finite size effects (as noted in [Valle-Pérez et al., 2018]), we cut off all frequencies less than or equal to 2.

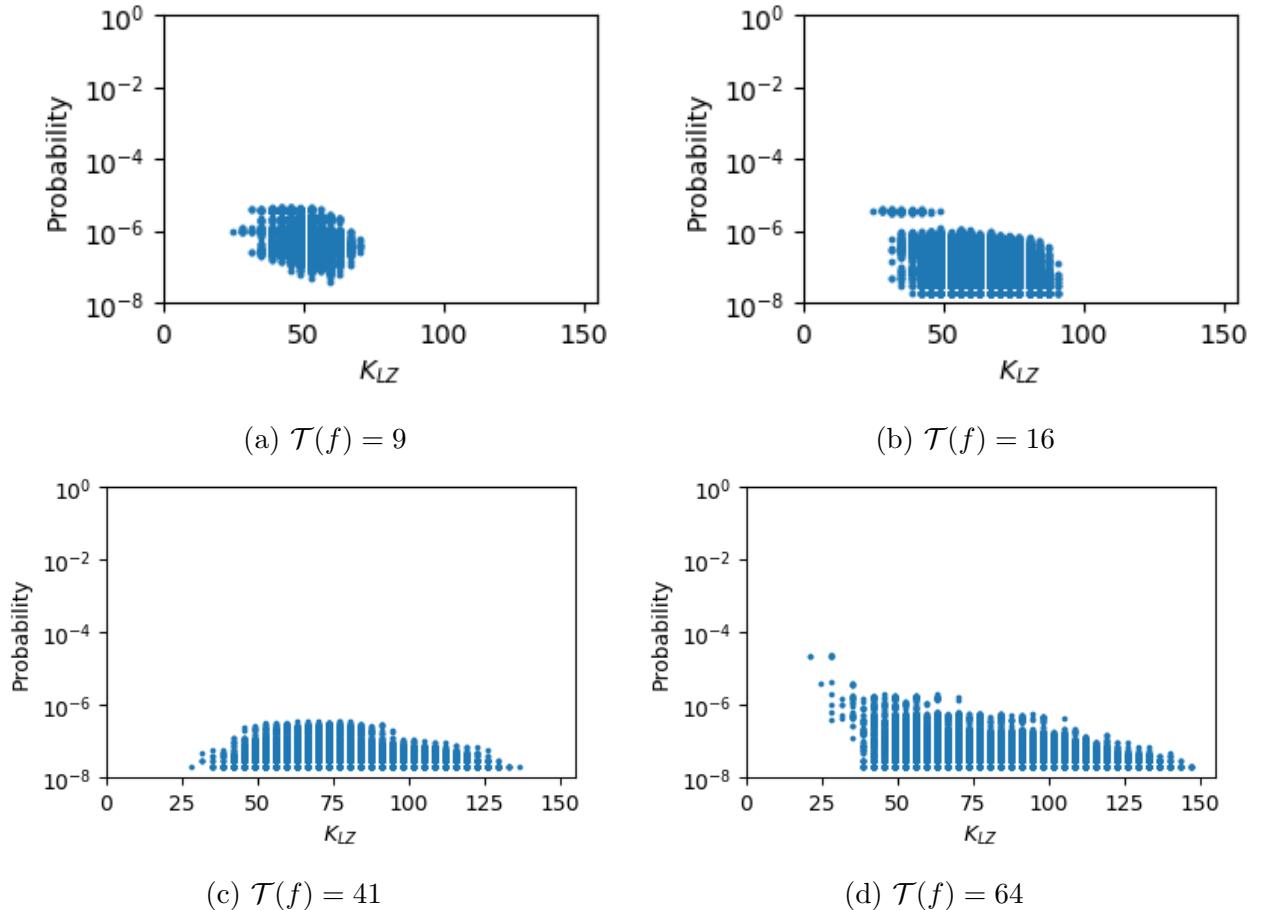


Figure E.4: $P(f)$ vs $K_{LZ}(f)$ at a selection of values of $\mathcal{T}(f)$, for a perceptron with input dimension 7, weights sampled from $\mathcal{N}(0, 1)$ and no threshold bias terms. We observe a large range in $K_{LZ}(f_t)$ for $f_t \in \mathbb{F}_t$ which increases as t approaches 64, which is to be expected – for example the function $f = 0101\dots$ is very simple and has maximum entropy, and we expect there to exist higher complexity functions at higher entropy. Consistent with the bound in [Valle-Pérez et al., 2018], simpler functions tend to have higher probabilities than more complex ones. The data-points at especially high probabilities in Figure E.4d correspond to the function $f = 0101\dots$ and equivalent strings after permuting dimensions.

As can be seen in Figure 3.1 the probability-complexity graph satisfies the simplicity bias bound Equation (2.1) for all functions. Now, in Figure E.4) we observe subsets \mathbb{F}_f of the overall set of functions. Firstly, we observe, as expected, that smaller t means a smaller range in K_{LZ} , since high complexity functions are not possible at low entropy. Conversely, low complexity functions are possible at high entropy (say for 010101...), and so a larger range of probabilities and complexities is observed for $t = 64$. For these larger entropies, an overall simplicity bias within the set of fixed t can be observed.

The larger range observed in $t = 64$ and $t = 16$ (compared to $t = 41$ and $t = 9$) can be explained by the presence of highly ordered functions having those t values - for example, in $t = 64$, there is $f = 0101\dots$ and its symmetries; and in $t = 16$ there are functions such as $f = 00010001\dots$ and its symmetries. The other two t values do not divide 2^7 , so there will be no functions with such low block entropy (implying low K_{LZ}).

We also demonstrate differences in the variation in $P(f)$ vs $K_{LZ}(f)$ when w is sampled from uniform distributions, in Figure E.5, and compare these plots to those in Figure E.4. Whilst we know from Theorem 3.3.1 that sampling w from a uniform distribution will not affect $P(t)$, it is not hard to see that there will be some variation in function probability within the classes \mathbb{F}_t . We observe that the simple functions which have high probability for $t = 64$ when the perceptron is initialised from a Gaussian (Figure E.4d) have lower probabilities in the uniform case (Figure E.5d). We comment further on this behaviour in Section E.2.2. However, we see limited differences in their respective rank-probability plots (Figure E.1a and Figure E.1d).

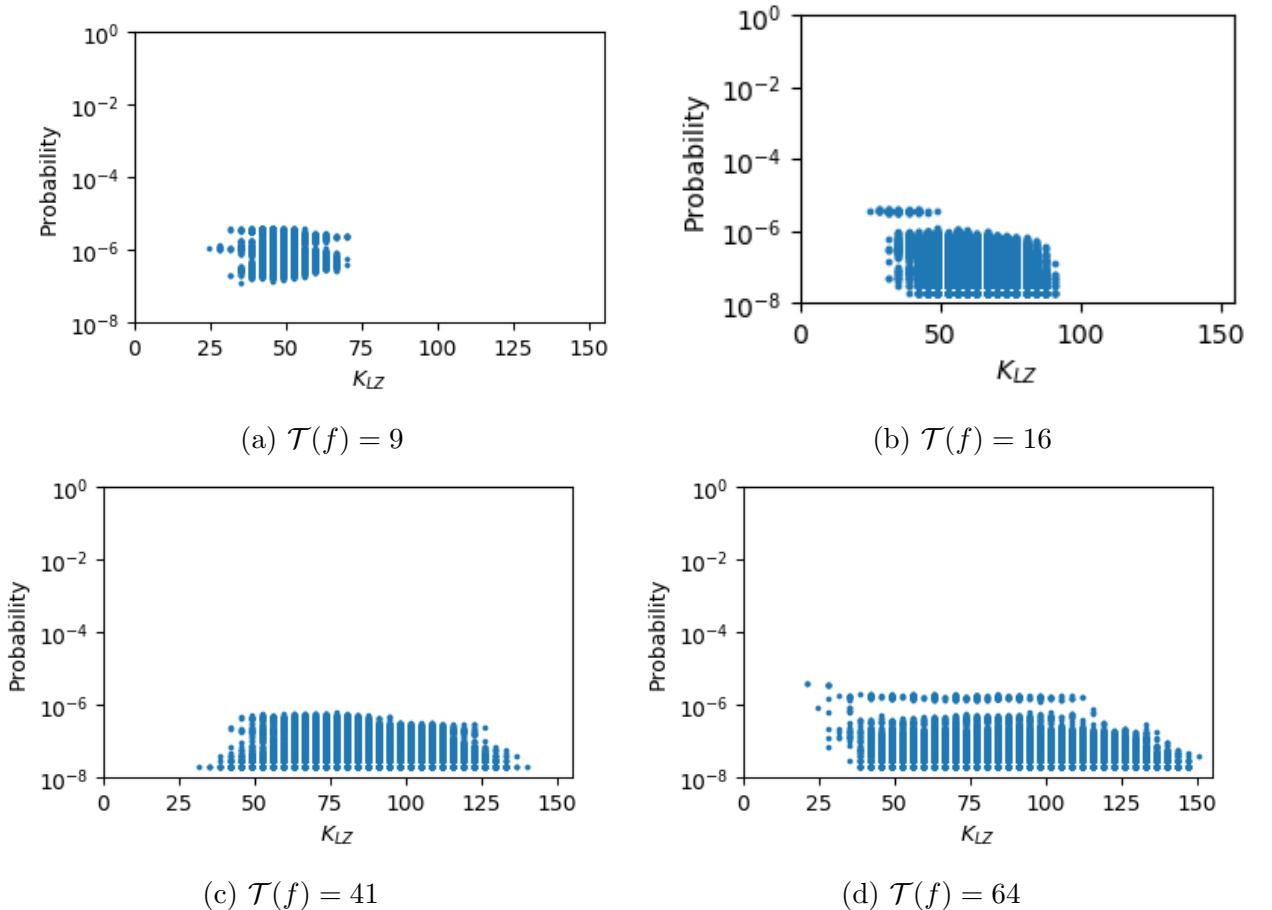


Figure E.5: $P(f)$ vs $K_{LZ}(f)$ at a selection of values of $\mathcal{T}(f)$, for a perceptron with input dimension 7, weights sampled from a centered uniform distribution and no threshold bias terms. We compare to Figure E.4, and observe that uniform sampling reduces slightly the simplicity bias within the sets \mathbb{F}_t (see Section E.2.2).

E.2.2 Substructure within $\{0, 1\}^n$

Consider a subset $\mathcal{H}^m \subset \{0, 1\}^n$ such that $0 \in \mathcal{H}^m$. We have $\binom{n}{m}$ such subsets. Then the marginal distribution over \mathcal{H}^m is given by

$$P\left(\sum_{x \in \mathcal{H}^m} \mathbf{1}(\langle w, x \rangle) = t\right) = 2^{-m} \quad (\text{E.5})$$

This is again independent of the distribution of w (provided it's symmetric about coordinate planes). We give two example applications of Equation (E.5) in Equation (E.6). We use $*$ to

mean any allowed value, and sum over all allowed values,

$$\begin{aligned} \sum_{*} P(f = "0 * 0 * \dots 0 * 0 *") &= 2^{-(n-1)} \\ \sum_{*} P(f = "0 * * * \dots 0 * * *") &= 2^{-(n-2)} \end{aligned} \quad (\text{E.6})$$

We can apply the same argument that we applied in Section 3.3 to any set of bits in f defined by some \mathcal{H}^m , to show that there is an “entropy bias” within each of these substrings. However, these identities imply a strong bias within each \mathbb{F}_t . For the case of full expressivity, assuming each value of t has probability 2^{-n} , and every string with the same value of t is equally likely, one gets probabilities very close to those in Equation (E.6) (although slightly lower). However, the perceptron is not fully expressive, so it is unclear how much the probabilities on Equation (E.6) are due purely to the entropy bias, and how much is due to bias within each \mathbb{F}_t .

It is difficult to calculate the exact probabilities of any function for Gaussian initialisation². It may not be possible to fine-grain probabilities analytically further than Equation (E.5) (although we can use the techniques in Section 3.3.3 to come up with analytic expressions for $P(f)$ for all f).

However, we can calculate some probabilities quite easily when w is sampled from a uniform distribution, $w \sim \mathcal{U}^n(-1, 1)$. By way of example, we calculate $P(f = \tilde{f})$ for $\tilde{f} = "0101 \dots 0101"$. The conditions for \tilde{f} are as follows: $w_n > 0$, $w_i < 0 \forall i \neq n$ and $\sum_j w_j > 0$, so

$$P(f = \tilde{f}) = 2^{-n} \int_0^1 \frac{x^{n-1}}{(n-1)!} dx = \frac{2^{-n}}{n!} \quad (\text{E.7})$$

Using Equation (3.2), we can calculate how much more likely the function is than expected,

$$\frac{P(f = \tilde{f})}{\langle P(f_{t=2^n-1}) \rangle} = \frac{|F_{t=2^n-1}|}{n!} \quad (\text{E.8})$$

For $n = 5$, we can calculate Equation (E.8) using³ $|F_{t=2^n-1}| = 370$ and $w \sim \mathcal{U}^n(-1, 1)$ and obtain $P(f = \tilde{f})/\langle P(f_{t=2^n-1}) \rangle = 3.08$. This clearly shows that \tilde{f} is significantly more likely

²Except for $f_{t<3}$, because we know all functions within constant t for $t < 3$ are equivalent under permutation of the dimensions so all their probabilities are equal to the average probabilities given by Equation (3.2)

³370 unique functions were obtained by sampling 10^{10} weight vectors so this is technically a lower bound

than expected just by using results from Equation (3.2). Empirical results further suggest that for Gaussian initialisation of w , $P(f = \tilde{f})/\langle P(f_{t=2^n-1}) \rangle \approx 10$. This, plus data in Section E.2.1 suggest that Gaussian initialisation may lead to more simplicity bias.

E.3 Towards understanding the simplicity bias observed within \mathbb{F}_t

Here we offer some intuitive arguments (primarily from Christ Mingard, and the author) that aim to explain why there should be further bias towards simpler functions within \mathbb{F}_t .

We first need several definitions.

We define a set $\mathbb{A} \subset R_{\geq 0}^n$ such that $a \in \mathbb{A}$ iff $a_i < a_j \forall i < j$, interpreted as the absolute values of the weight vector.

We now define four sets, Γ Σ and Υ , which classify the types of linear conditions on the weights of a perceptron acting on an n -dimensional hypercube:

1. Γ , the set of permutations in $\{1, \dots, n\}$ (which we can interpret as permutations of the axes in \mathbb{R}^n or as possible orders of the absolute values of the weights if no two of them are equal).
2. $\Sigma = \{-1, +1\}^n$ (which is interpreted as the signature of the weight vector)
3. Υ is the set of linear inequality conditions on components of $a \in \mathbb{A}$, which include more than two elements of a (so they exclude the conditions defining \mathbb{A}).

A unique weight vector can be specified giving its signature $\sigma \in \Sigma$, the order of its absolute values $\gamma \in \Gamma$, and a value of $a \in \mathbb{A}$. On the other hand, a unique *function* can be specified by a set of linear conditions on the weight vector w . These conditions can be divided into three types: Σ (signs), Γ (ordering), and Υ (any other condition).

The intuition to understand the variation in complexity and probability over different functions is the following. Each function corresponds to a unique set of necessary and sufficient conditions on the weight vector (corresponding to the faces of the cone in weight space producing that function). We argue that different functions have conditions which vary a lot in complexity,

and we conjecture that this correlates with their probability, as we discuss in Section 3.3.3 in the Chapter 3.

As a first approach in understanding this, we consider the role of symmetries under permutations of dimensions. Any string that is symmetric under a permutation of k dimensions⁴, can't have necessary conditions that represent relative orderings of those k dimensions. Furthermore the set of conditions must be invariant under these permutations. This strongly constraints the sets of conditions that highly symmetric strings like "010101..." or "1111...00000..." can have, to be relatively simple sets.

We now consider the set of necessary conditions in Υ for different functions. We expect that conditions in Υ are more complex than those in Γ and Σ . Furthermore, we find that functions have a similar number of minimal conditions⁵, so that more conditions in Υ seems to imply fewer conditions in Γ and Σ , and therefore, a more complex set of conditions overall.

We are going to explicitly study the functions within each set of constant t , for some small values of t , and find their corresponding conditions in Υ and Σ . We fix Γ to be the identity for simplicity. The conditions for a particular function should include the conditions we find here plus their corresponding conditions under any permutation of the axes which leaves the function unchanged. This means that on top of the describing the conditions for a fixed Γ , we would need to describe the set of axes which can be permuted. This will result in a small change to the Kolmogorov complexity of the set of conditions, specially small for functions with many symmetries or very few symmetries.

We find that the conditions appear to be arranged in a decision tree (a consequence of Theorem 3.3.1) with a particular form. First, we will prove the following simple lemma which bounds the value of t that is possible for some special cases of σ .

Lemma E.3.1. *We define $t_{\min}(\sigma)$ to be the minimum possible value of $\mathcal{T}(\sigma \odot a)$ for fixed σ over all a . We define t_{\max} similarly. Consider $\sigma = (\underbrace{-1, \dots, -1}_{k-1}, +1, \underbrace{-1, \dots, -1}_{n-k})$. Then:*

1. $t_{\max}(\sigma) = 2^{k-1}$

2. $t_{\min}(\sigma) = k$

⁴or example, "0101010..." is symmetric under permutation of $n - 1$ dimensions

⁵In fact we conjecture that the number of conditions is close to n for most functions, although we empirically find that it can sometimes be larger too

3. *Changing $\sigma_i = \{+1\}$ for some $i < k$ will lead to an increase in $t_{min}(\sigma)$*

Proof. 1. For any a , $f(x) = 1$ for all (exactly k) points $x \in \{0, 1\}^n$ which satisfy

$$(x_k = 1) \text{ and } (x_i = 1 \text{ for exactly one } i \leq k) \text{ and } (x_j = 0 \text{ if } j \neq i, k)$$

We can set $f(x) = 0$ for all other $x \in \{0, 1\}^n$ by imposing the condition $a_1 + a_2 > a_k$. Thus $t_{min}(\sigma) = k$.

2. We can restrict the values of a_i for $i < k$ to be arbitrarily smaller than a_k provided they satisfy the ordering condition on a , and thus if we impose the condition

$$\sum_{i=1}^{i=k-1} a_i < a_k$$

then for any a , $f(x) = 1$ for all $x \in \{0, 1\}^n$ which satisfy

$$(x_k = 1) \text{ and } (x_l = 0 \text{ if } l > k)$$

We can see that, for all a , $f(x) = 0$ for any x which does not satisfy these conditions, because $a_k < a_l$ for all $k < l$ and σ_k is the only positive element in σ . Thus $t_{max} = 2^{k-1}$

3. On changing σ such that $\sigma_i = \{+1\}$, all $x \in \{0, 1\}^n$ which previously satisfied $f(x) = 1$ will remain mapped to 1, plus at least the one-hot vector with $a_i = 1$. \square

We will now sketch a procedure that allows one to enumerate the conditions in Υ and Σ (corresponding to conditions on a and on σ respectively) such that they satisfy $\mathcal{T}(\sigma \odot a) = t$, for any given t .

1. From Theorem E.3.1 we see that all $\sigma_i = -1$ for $i > t$ in order for $t_{min}(\sigma) \leq t$.
2. Iterate through all 2^t distinct σ which satisfy 1., and retain only those which also satisfy $t_{min}(\sigma) \leq t$. $t_{min}(\sigma)$ can be computed by counting the number of inputs $x \in \{0, 1\}^n$ such that for every $x_i = 1$ with $\sigma_i = -1$, there exists a x_j with $j > i$ such that $\sigma_j = 1$. These are all the x such that σ and a imply they are mapped to 1 without further conditions on a .

3. Find conditions on these σ such that $\mathcal{T}(\sigma \odot a) = t$. We can find the possible values of $\mathcal{T}(\sigma \odot a)$ by first ordering the x in a way that satisfies $x < x'$ if x has less 1s than x' . We then traverse the decision tree corresponding to mapping each of the x , in order, to either 1 or 0. At each step, we propagate the decision by finding all x not yet assigned an output, which can be constructed as a linear combination of x 's with assignments, where the coefficients in the combination have opposite signs for x' mapped to different outputs. We stop the traversal if at some point more than t points are mapped to 1. Each of the decisions in the tree, correspond to a new condition on a . We denote these conditions by v .

For small values of t one can perform this search by hand. For example, we consider $t = 4$. We find that all signatures with $\sigma_{i>4} = -1$ are the only set that have $t_{\min} \leq 4$. As an example we consider the signature $\sigma = \{+1, +1, -1, \dots, -1\}$. For this signature to result in $\mathcal{T}(\sigma \odot a) = 4$, we need conditions on $x_1 = (1, 1, 0, 1 \dots)$ and $x_2 = (1, 1, 1, 0 \dots)$ which are $(a_4 > a_1 + a_2)$ and $(a_3 < a_1 + a_2)$. Figure E.6 shows the full sets for $t = 4$ and $t = 5$. We observe that each branching corresponds to complementary conditions - which is to be expected, as there exists a signature producing t for any a , as per Theorem 3.3.1.

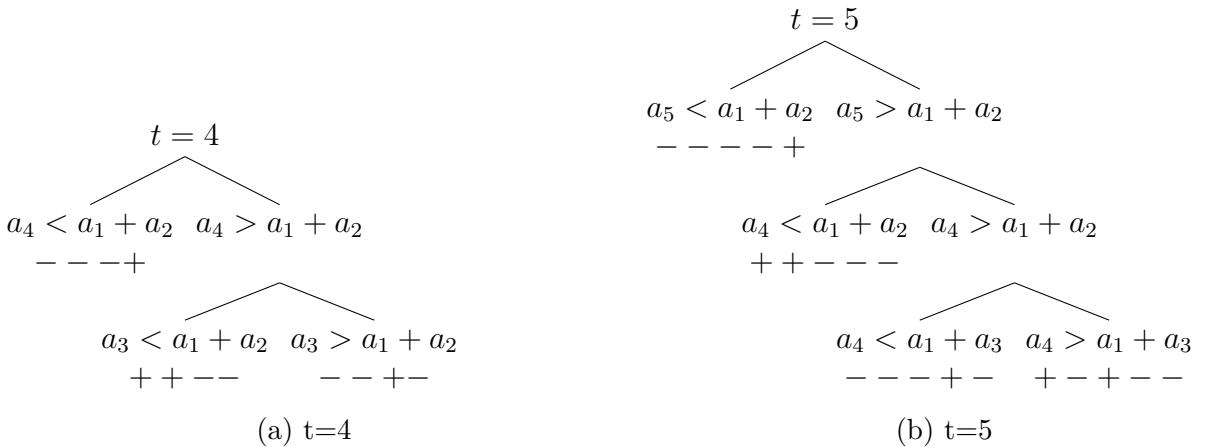


Figure E.6: We assume that the signs to the right of those shown are all negative. We can list the various non-equivalent classes of functions and their conditions in a pictorial form. A condition at any node in the graph is also a condition for any daughter node - by way of example, we would read the conditions of $t = 4$ for the sign arrangement $+ + --$ (see Figure E.6a) as $((a_4 > a_1 + a_2) \cap (a_3 < a_1 + a_2))$.

Each distinct condition on a and σ produces a unique function f , as the constructive procedure above produces the inequalities on a_i by specifying the outputs of each input not already implied by the set of conditions, thus uniquely specifying the output of every input. We now show that there is a large range in the number of conditions in Υ required to specify different function. First, as the analysis in the proof of Theorem E.3.1 shows, for every t there exists a signature which only requires one further condition,

$$\sigma = (\underbrace{-1, \dots, -1}_{t-1}, +1, \underbrace{-1, \dots, -1}_{n-t}), v = a_t < a_1 + a_2$$

Furthermore, we prove in Theorem E.3.2 that for all n there is at least one function which has $n - 2$ conditions in Υ (which are neither in Σ or Γ). This implies that there is a large range in the number of conditions in Υ for large n .

Lemma E.3.2. *There exists a function with $n - 2$ inequalities in Υ , that is not including those induced by Σ or in Γ .*

Proof. We prove this by induction.

Base case $n = 2$: There exists a function with minimal conditions corresponding to $(1, 0)$ mapping to 1 and $(1, 1)$ mapping to 0.

Assume that in n dimensions, there exists a function with n minimal conditions corresponding to points $\{p_i\}_{i=1}^n$ with $p_i = (1, \dots, 1, 0, \dots, 0)$ with 1s in the first i positions, and 0 in the last $n - i$ positions, being mapped to $f(p_i) = (1 + (-1)^{i+1})/2$. Now, in $n + 1$ dimensions, we can extend each of those n conditions (planes) by adding a 0 to the n th coordinate of each point p_i . We now consider a cone in n dimensions bounded by these planes and bounded by the $w_{n+1} = 0$ plane, with $w_{n+1} > 0$ if n is even or $w_{n+1} < 0$ if n is odd. If n is even, we can keep increasing w_{n+1} until we cross the plane $p_{n+1} = 0$, with $p_{n+1} = (1, \dots, 1)$. We do the same, decreasing w_{n+1} if n is odd. After we crossed the $p_{n+1} = 0$ plane, the boundaries of the plane will be $\{p_i\}_{i=1}^{n+1}$ with $p_i = (1, \dots, 1, 0, dots, 0)$ with 1s in the first i positions, and 0 in the last $n + 1 - i$ positions, finishing the induction.

□

We conjecture that more conditions in Υ (i.e. more complex conditions) correlates with

lower probability (if w is sampled from a Gaussian distribution). If this is true for higher t , we should expect to see high probabilities correlating with lower complexities, which if true would explain the “simplicity bias” we observe beyond the entropy bias.

Appendix F

Further results on PAC-Bayes learning curves

In this appendix we present further results from the experiments in Chapter 5.

F.1 Learning curves versus dataset for all resnets

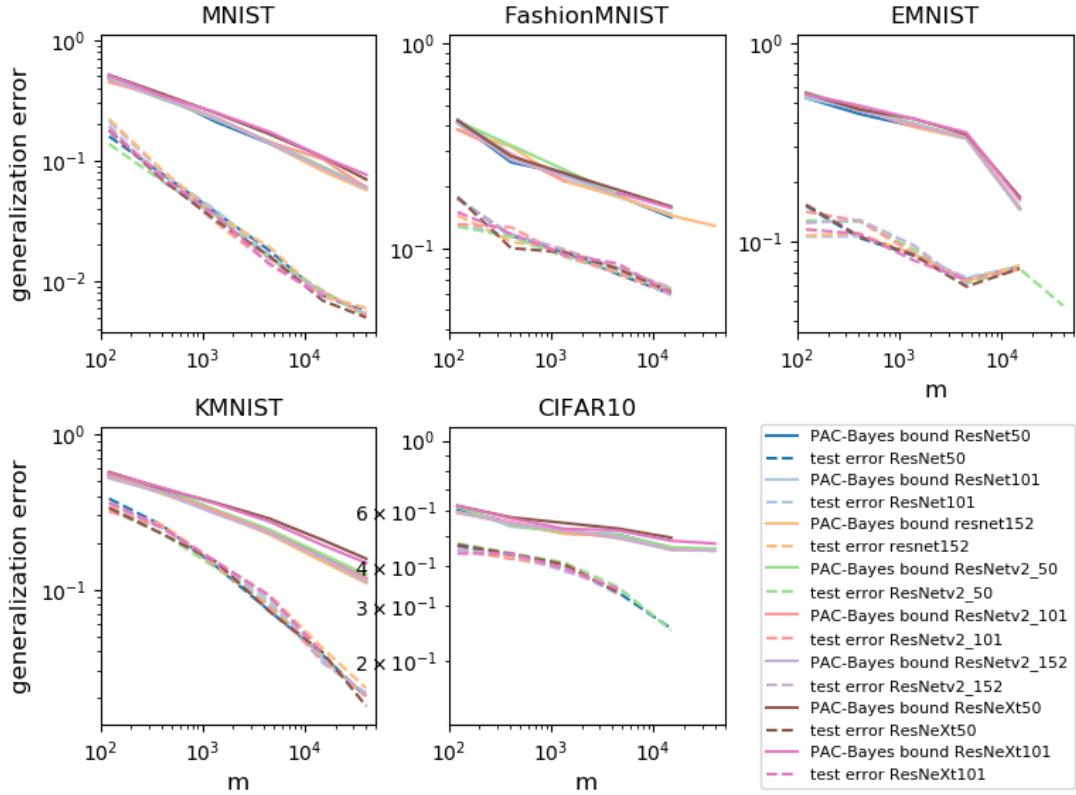


Figure F.1: **Comparing different resnet architectures.** Learning curves for the test error and the PAC-Bayes bounds for different resnet architectures for different datasets. The DNNs were trained using Adam and batch size 32 to 0 training error.

F.2 Learning curves versus dataset for all densenets

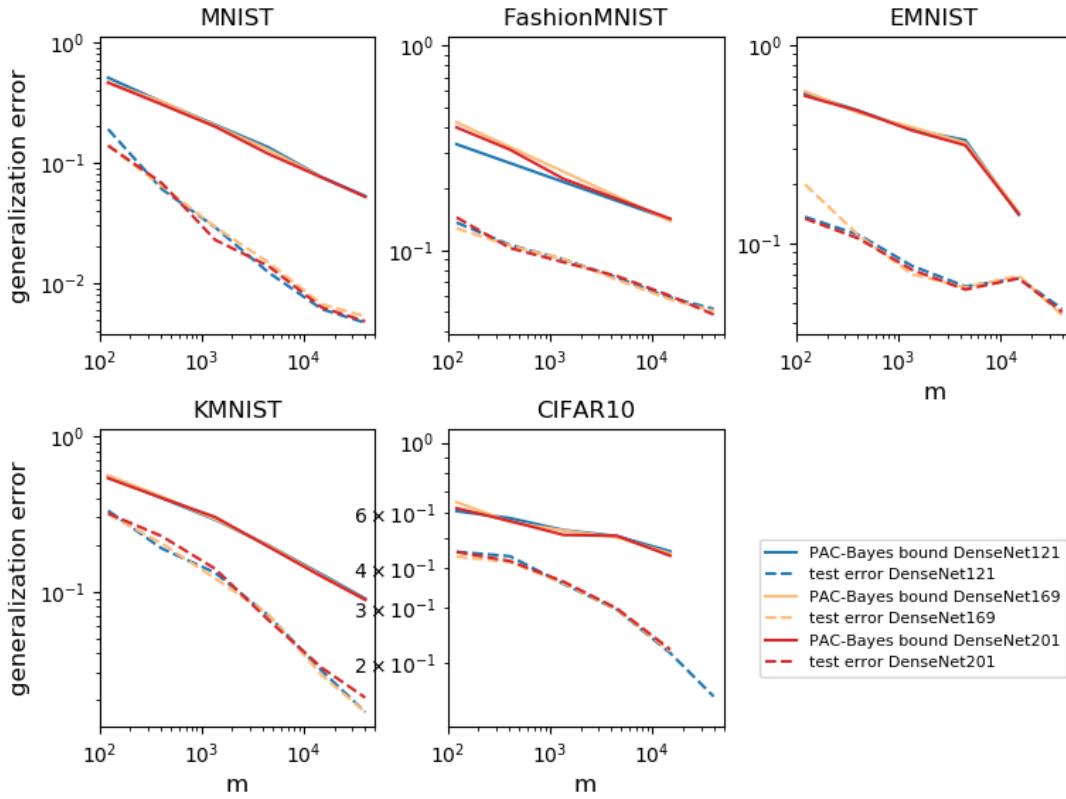


Figure F.2: **Comparing different densenet architectures.** Learning curves for the empirical test error and the PAC-Bayes bounds for different densenet architectures for different datasets. The DNNs were trained using Adam and batch size 32 to 0 training error.

F.3 Error versus bound for batch size 32 for more datasets

Here we show the bound vs error for the rest of architectures not shown in Section 5.0.3.

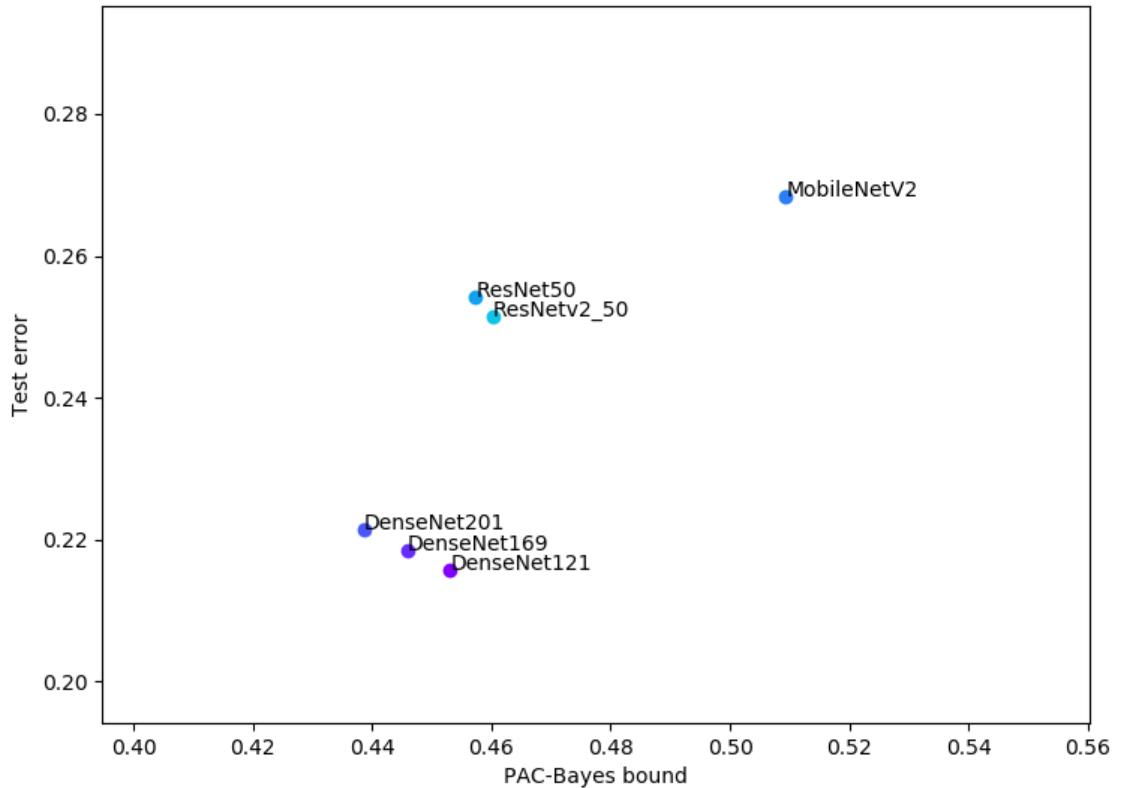


Figure F.3: PAC-Bayes bound versus test error for different models trained on a sample from CIFAR of size 15k, with batch size 32. FCN and CNNs are removed for clarity as they often have the relatively extreme values of test error and/or bound.

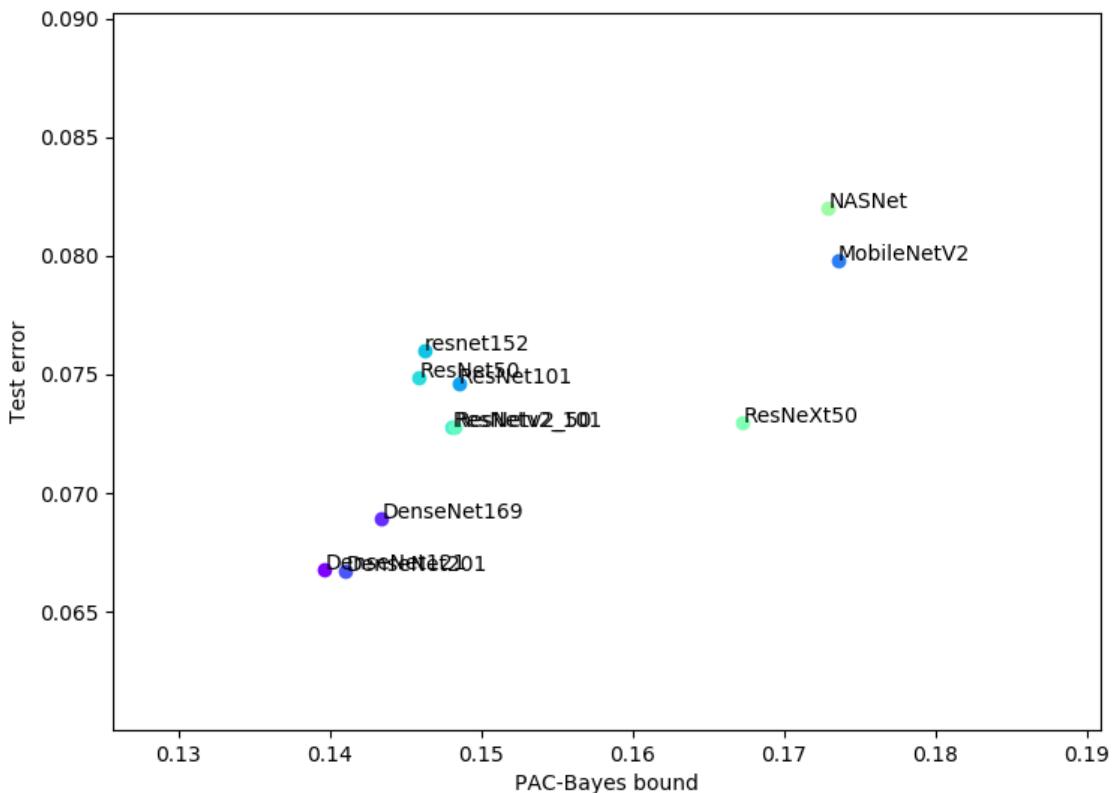


Figure F.4: PAC-Bayes bound versus test error for different models trained on a sample from EMNIST of size 15k, with batch size 32. FCN and CNNs are removed for clarity as they often have relatively extreme values of test error and/or bound.

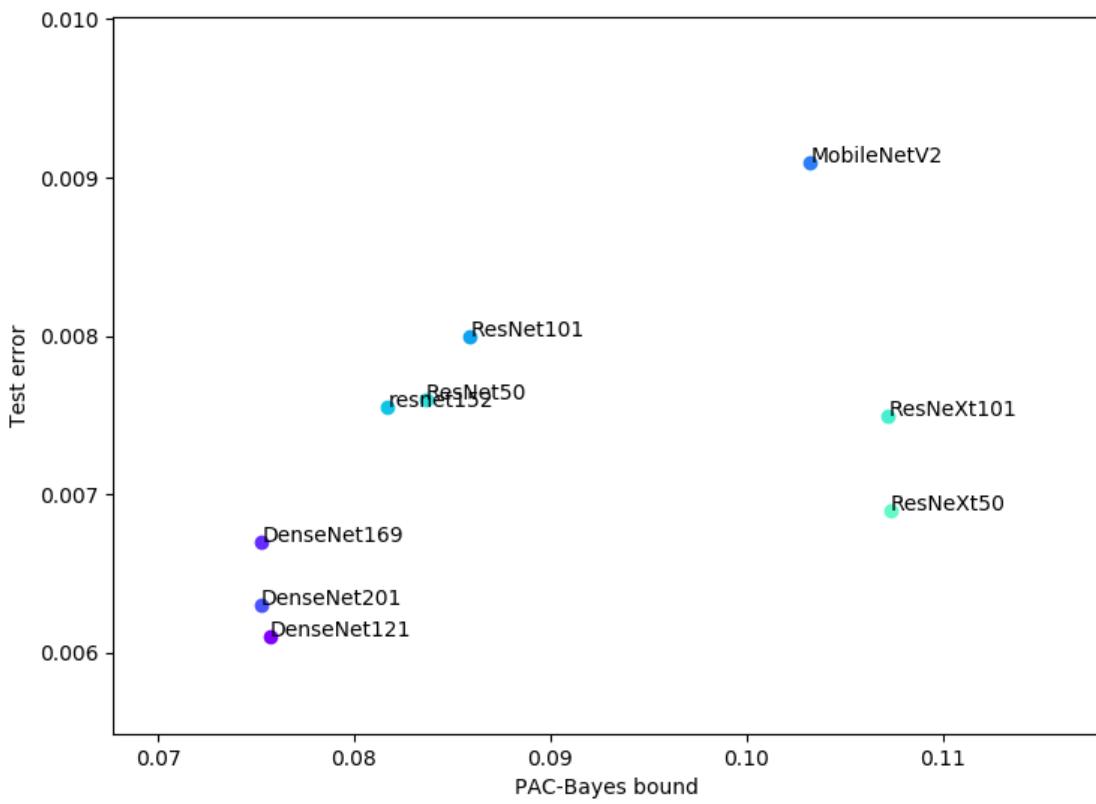


Figure F.5: PAC-Bayes bound versus test error for different models trained on a sample from MNIST of size 15k, with batch size 32. FCN and CNNs are removed for clarity as they often have relatively extreme values of test error and/or bound.

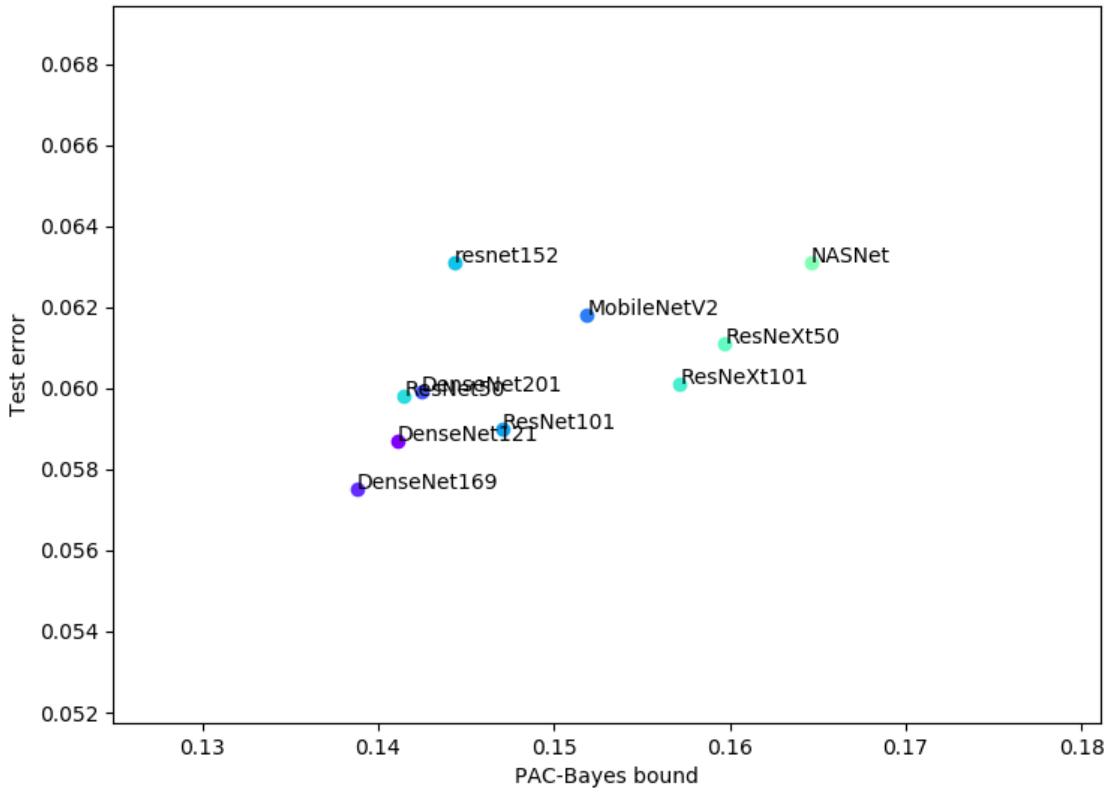


Figure F.6: PAC-Bayes bound versus test error for different models trained on a sample from Fashion-MNIST of size 15k, with batch size 32. FCN and CNNs are removed for clarity as they often have relatively extreme values of test error and/or bound.

F.4 Learning curves versus dataset for all architectures

Here we show all the learning curves for each architecture. This is the same data as in Section 5.0.2, but plotted in a different way.

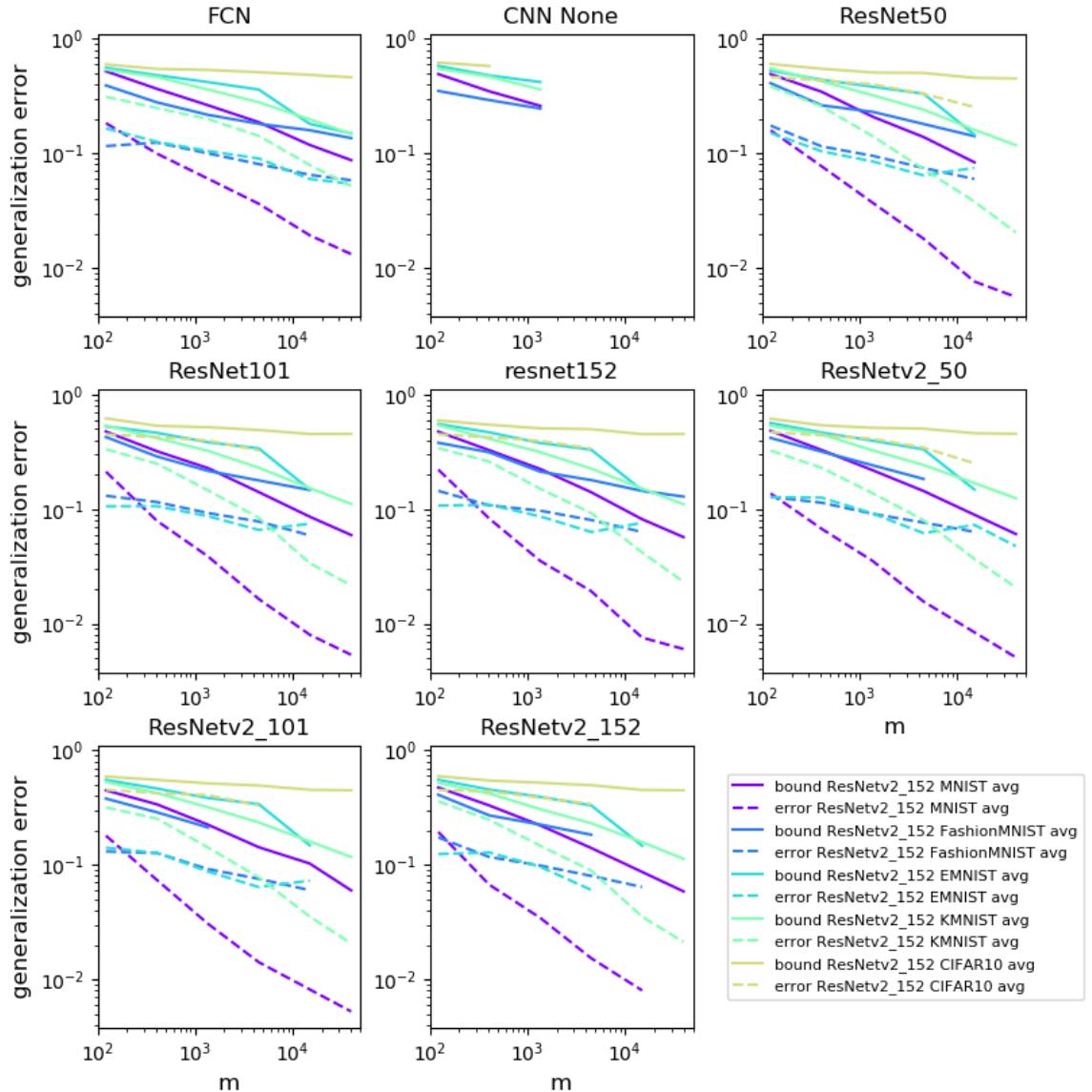


Figure F.7: Learning curves for the test error and the PAC-Bayes bounds for different architectures and for different datasets. The DNNs were trained using Adam and batch size 32 to 0 training error.

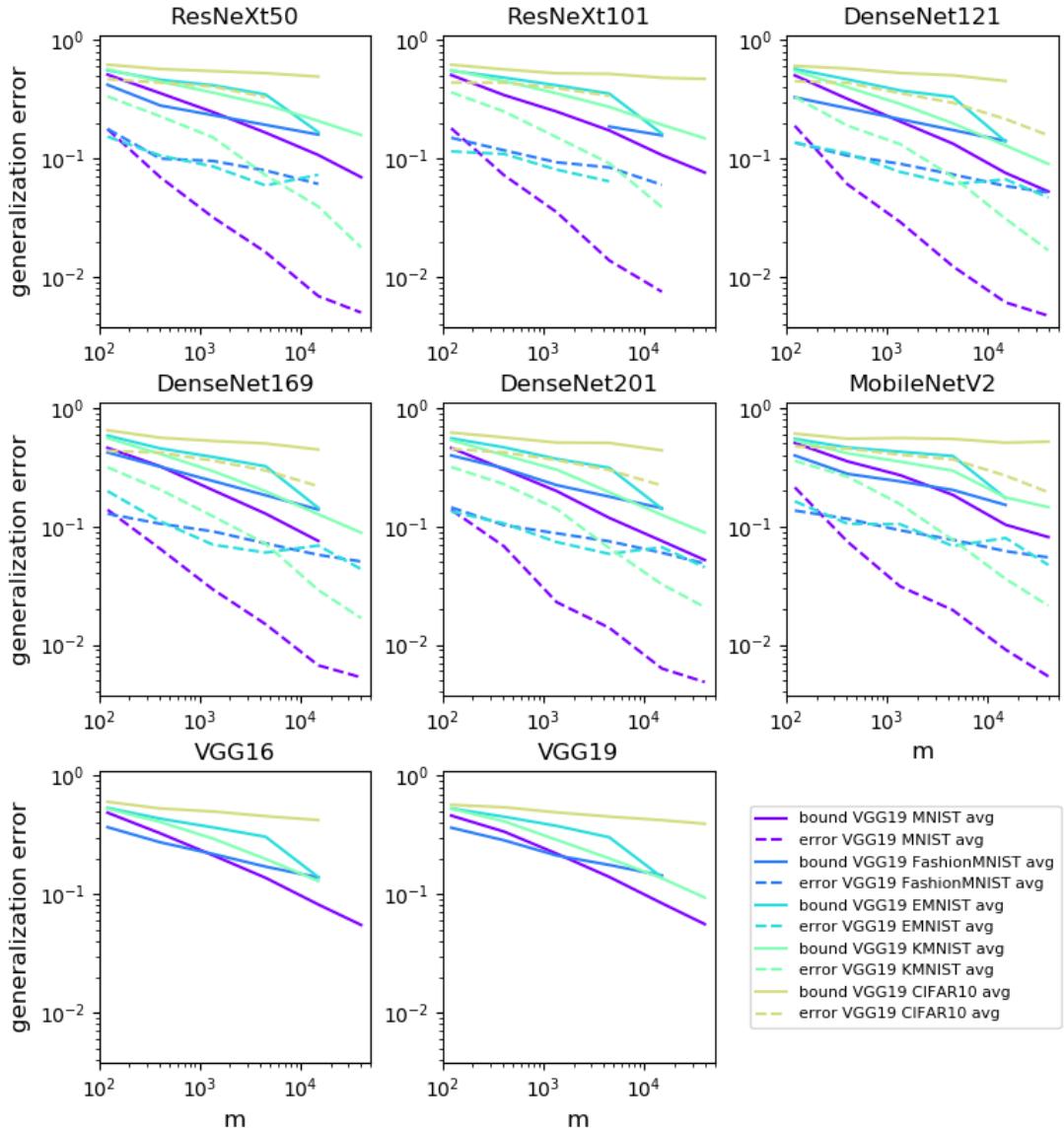


Figure F.8: Learning curves for the test error and the PAC-Bayes bounds for different architectures and for different datasets. The DNNs were trained using Adam and batch size 32 to 0 training error.

F.5 Learning curves for batch size 256

In this section, we show the results of the same set experiments as in Section 5.0.2, but performed with batch size 256. Note that in some cases we have more or less data relative

for batch 32, depending on which experiments could finish within our compute budget. We observe qualitatively similar results as for batch 32.

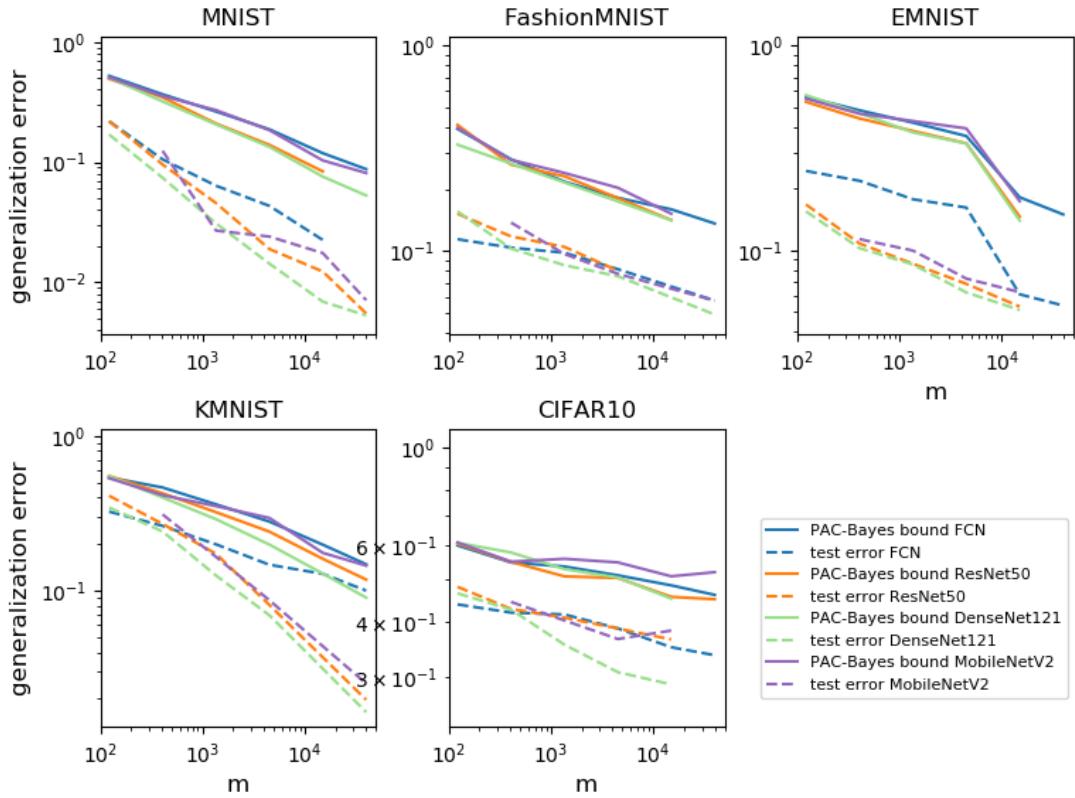


Figure F.9: Learning curves for the test error and the PAC-Bayes bounds for representative architectures for different datasets. The DNNs were trained using Adam and batch size 256 to 0 training error. We see the different architectures show similar learning curve power law exponent, which is matched closely by the PAC-Bayes bound.

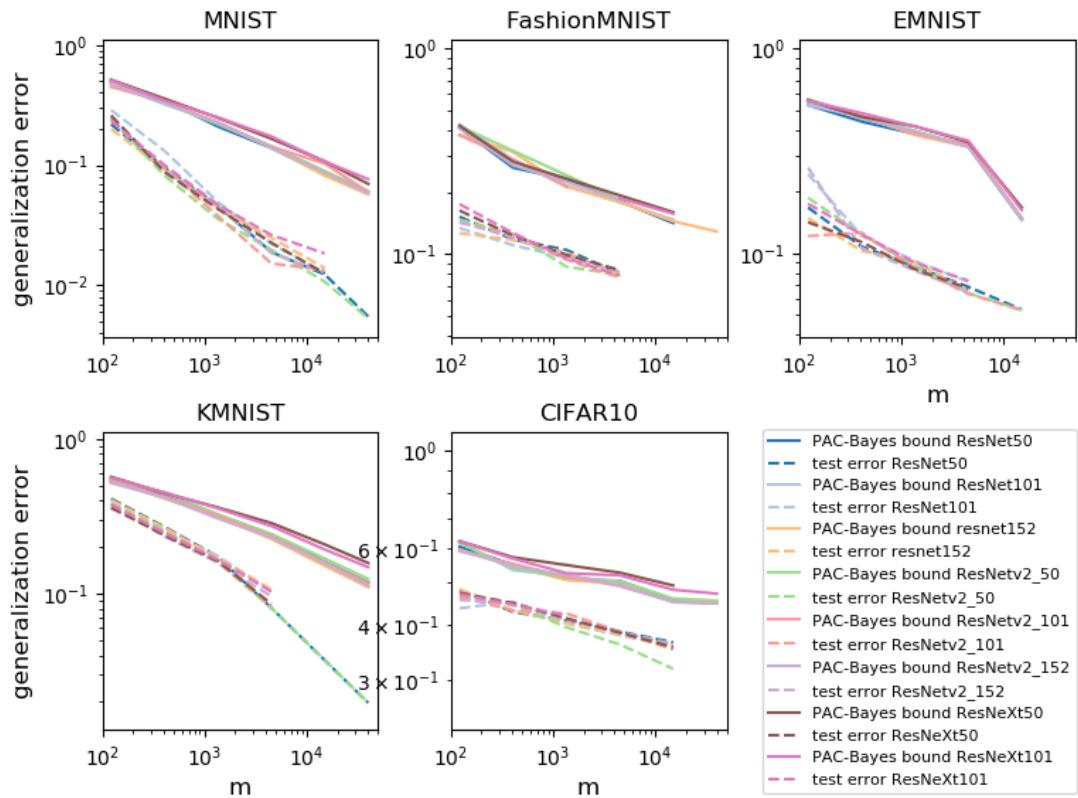


Figure F.10: Learning curves for the test error and the PAC-Bayes bounds for different resnet architectures for different datasets. The DNNs were trained using Adam and batch size 256 to 0 training error.

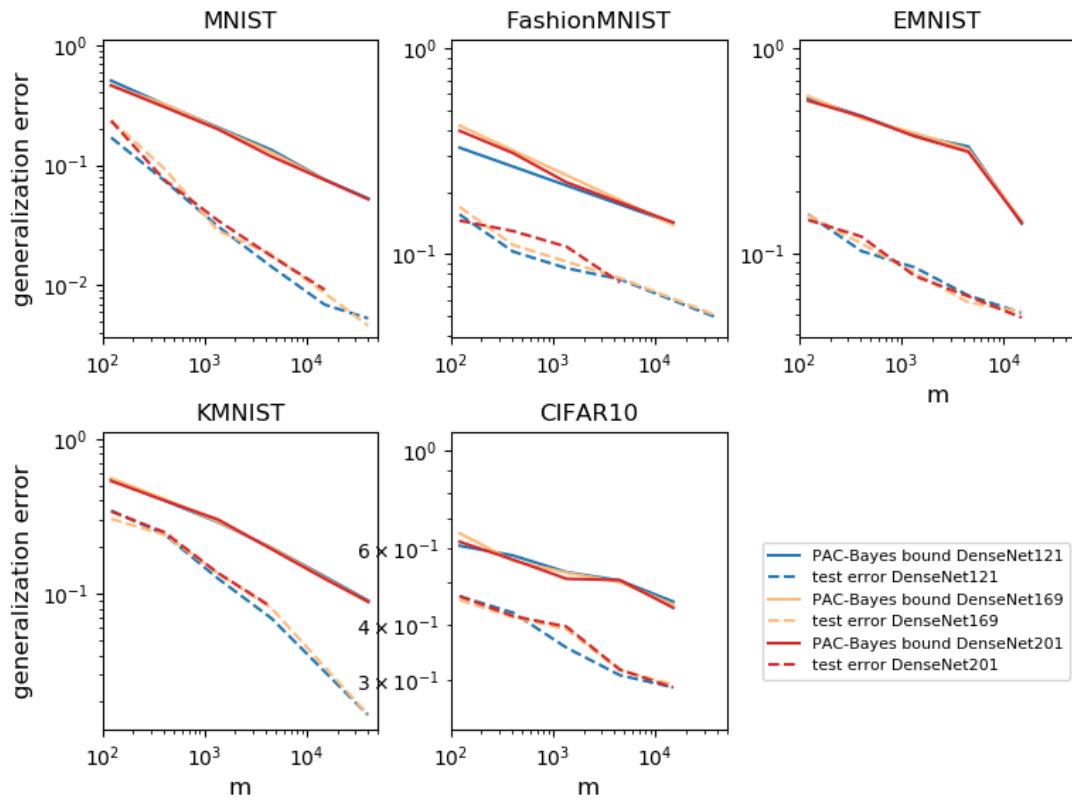


Figure F.11: Learning curves for the empirical test error and the PAC-Bayes bounds for different densenet architectures for different datasets. The DNNs were trained using Adam and batch size 256 to 0 training error.

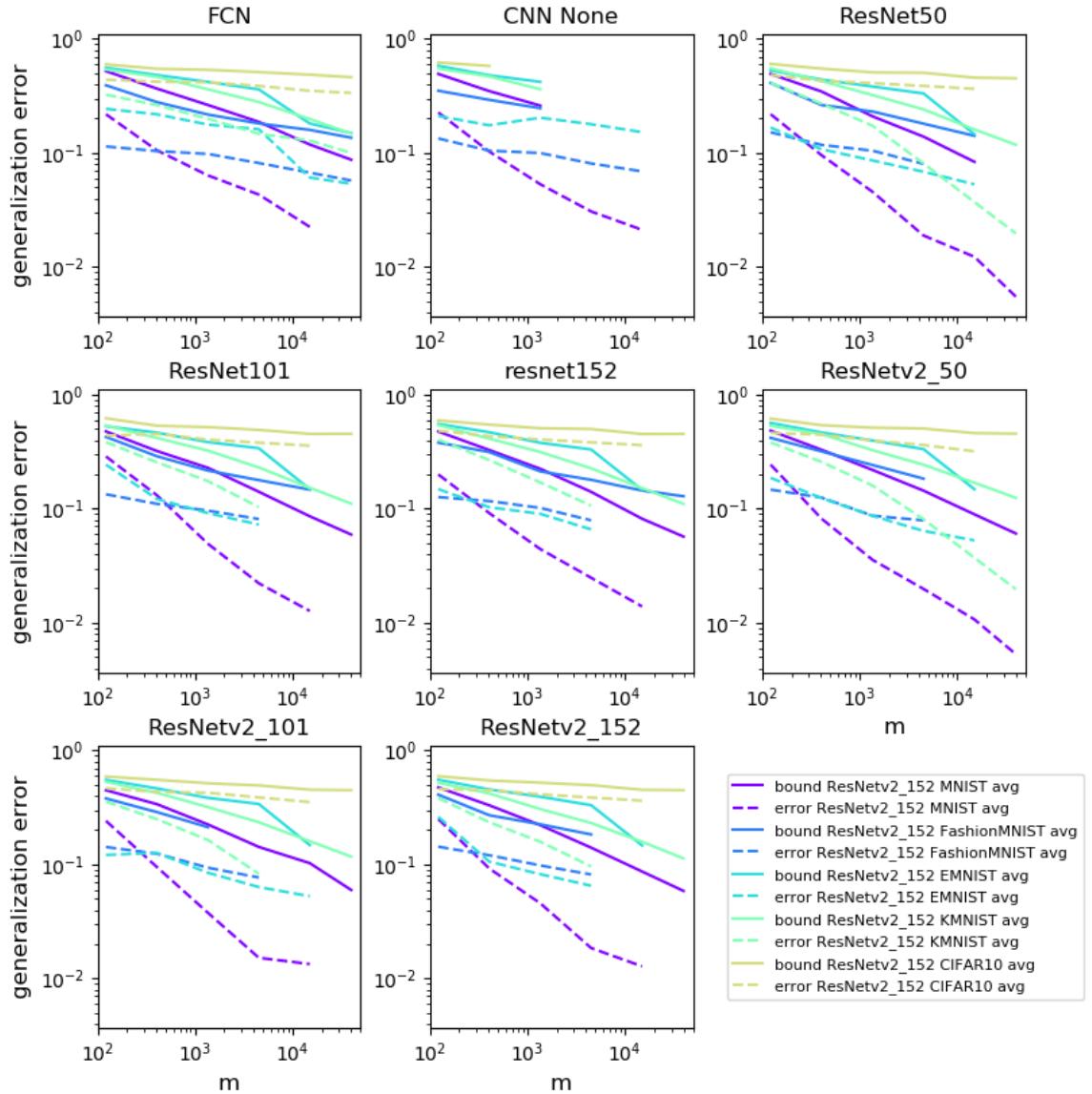


Figure F.12: Learning curves for the test error and the PAC-Bayes bounds for different architectures and for different datasets. The DNNs were trained using Adam and batch size 256 to 0 training error.

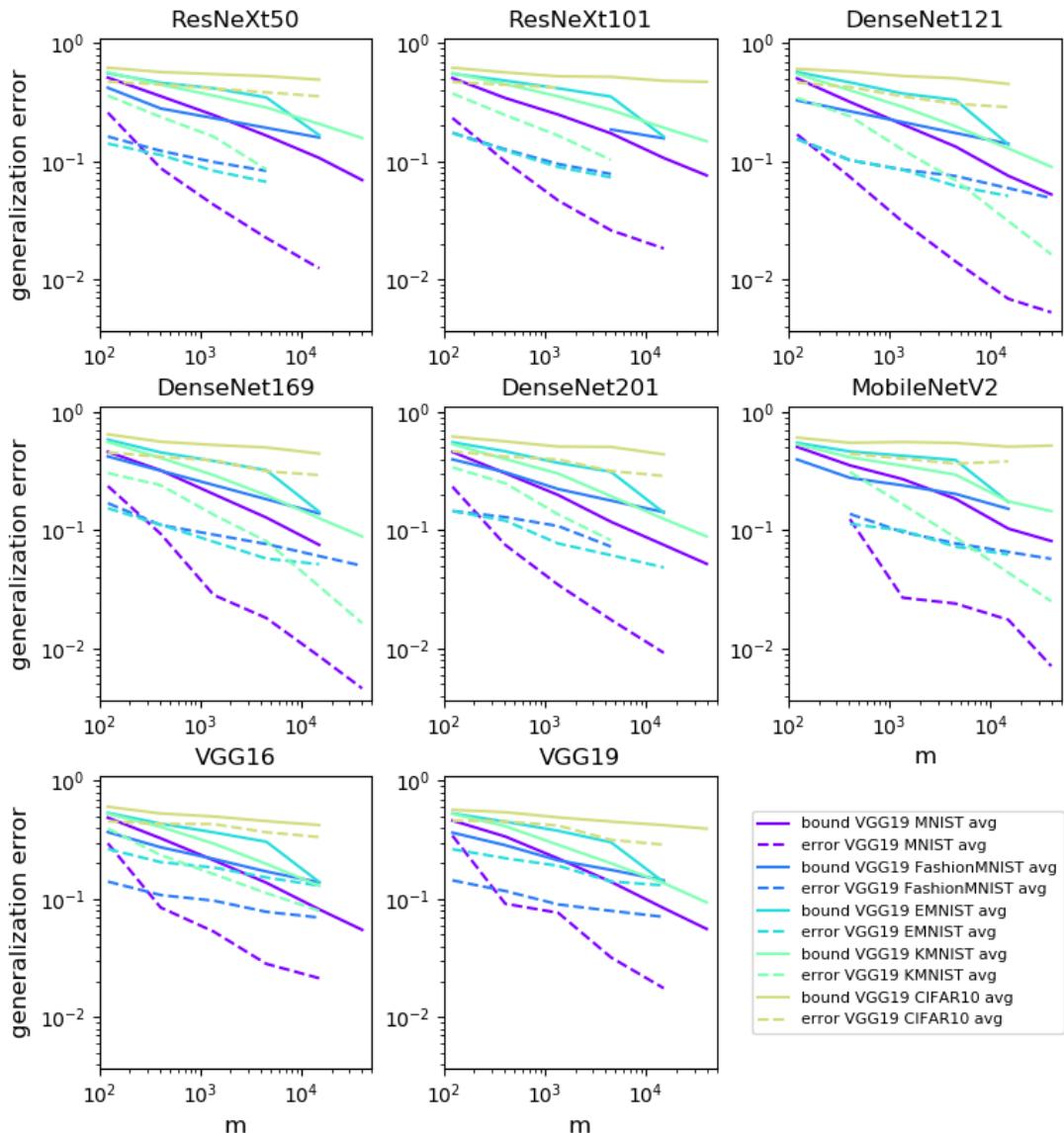
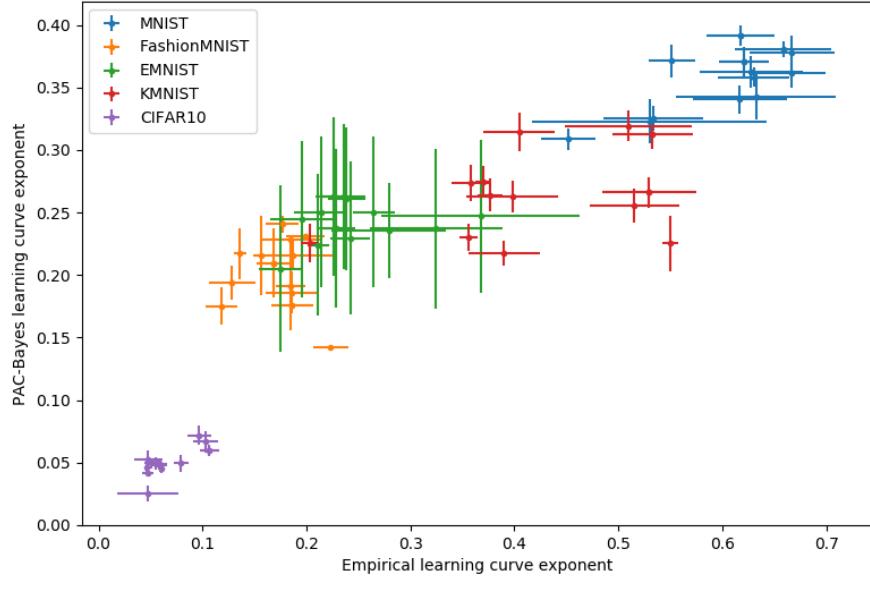
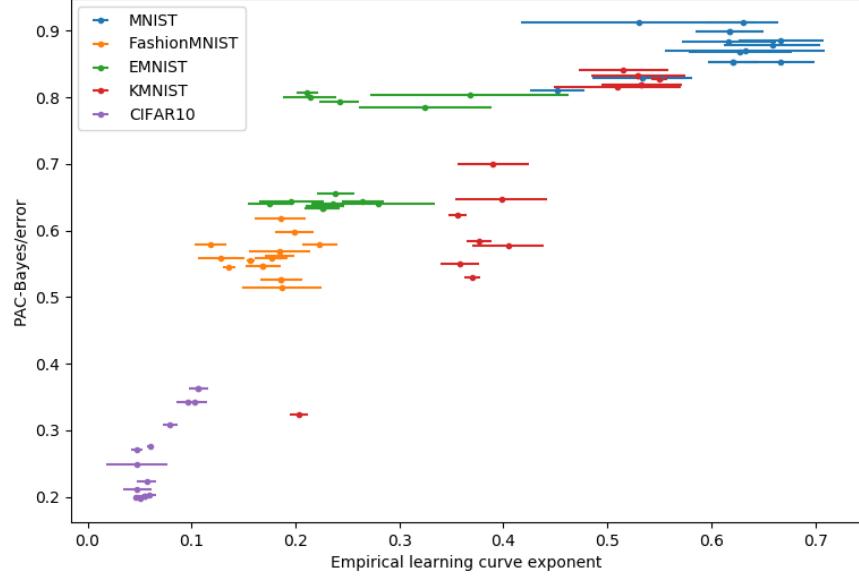


Figure F.13: Learning curves for the test error and the PAC-Bayes bounds for different architectures and for different datasets. The DNNs were trained using Adam and batch size 256 to 0 training error.



(a)



(b)

Figure F.14: Learning curve exponents of test error vs learning curve exponent estimated from PAC-Bayes bound for batch 256. In both plots, each marker shows the learning curve exponent, α obtained from a linear fit to either the learning curve corresponding to $\log \epsilon$ vs $\log m$ or the estimated exponent from the PAC-Bayes bound, for all the architectures and datasets in Section A.2. **(a)** The exponent is estimated from a linear fit to the log of the PAC-Bayes bound vs $\log m$. **(b)** The exponent is estimated as $1 - 1/C'$ where C' is the ratio of the PAC-Bayes bound and the error (see Theorem 4.5.2) The error bars are estimated standard errors from the linear fits. For the ratio estimate the errors due to fluctuations in dataset are negligible. Note that the errors do not take into account errors due to the EP approximation. Note that the exponents cluster according to dataset. The outliers for MNIST and KMNIST are both the FCN. The DNNs were trained using Adam and batch size 256 to 0 training error.

F.6 Error versus bound for batch size 256

In this section, we show the same experiments plotting the bound versus test error, as in Section 5.0.3, but for batch size 256.

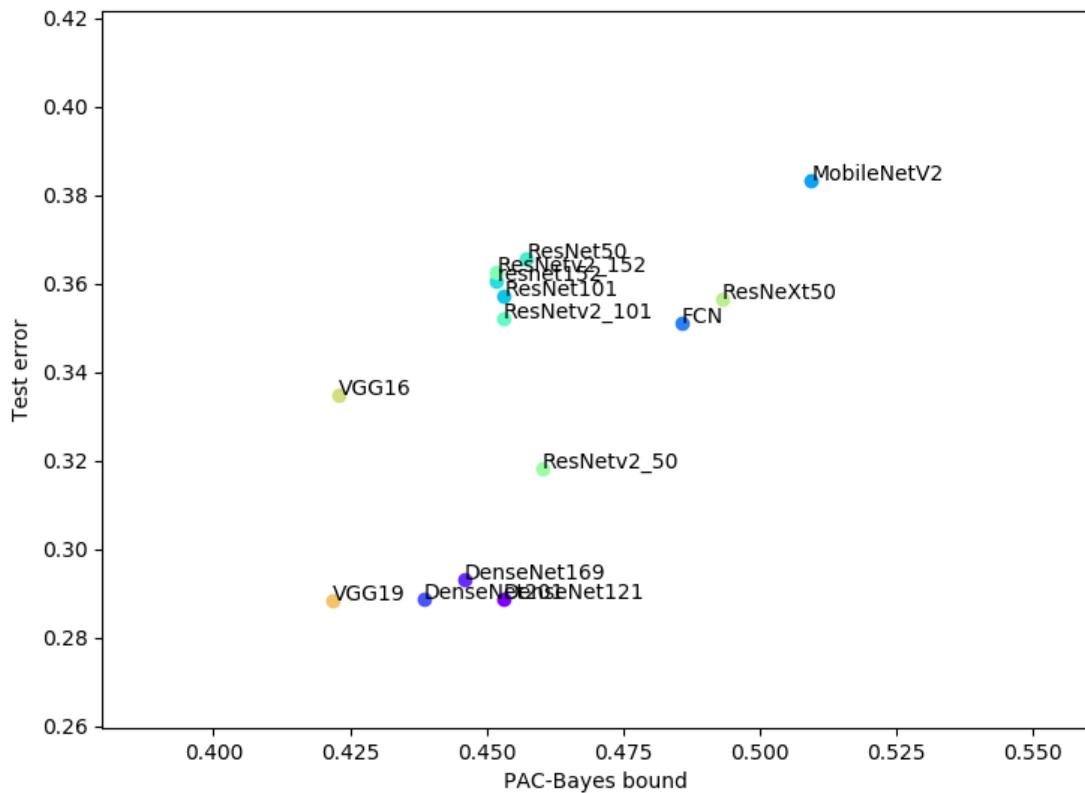


Figure F.15: PAC-Bayes bound versus test error for different models trained on a sample from CIFAR of size 15k, with batch size 256. CNNs are removed for clarity as they often have relatively extreme values of test error and/or bound.

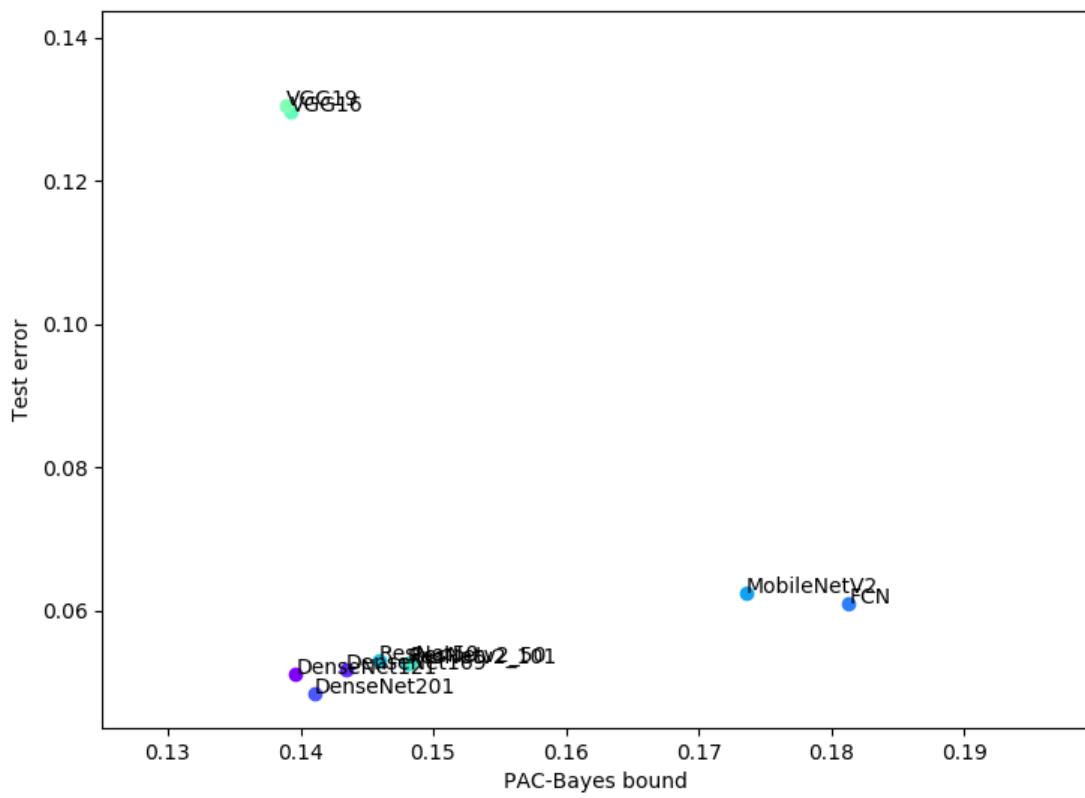


Figure F.16: PAC-Bayes bound versus test error for different models trained on a sample from EMNIST of size 15k, with batch size 256. CNNs are removed for clarity as they often have relatively extreme values of test error and/or bound.

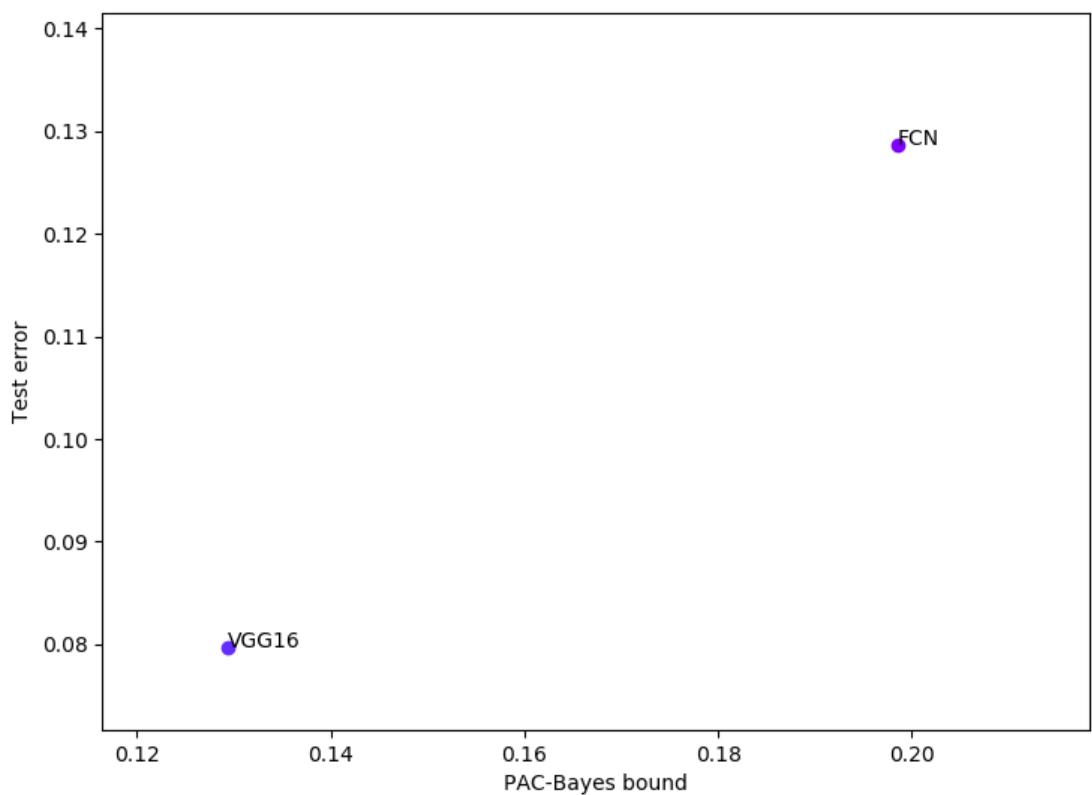


Figure F.17: PAC-Bayes bound versus test error for different models trained on a sample from KMNIST of size 15k, with batch size 256. CNNs are removed for clarity as they often have relatively extreme values of test error and/or bound.

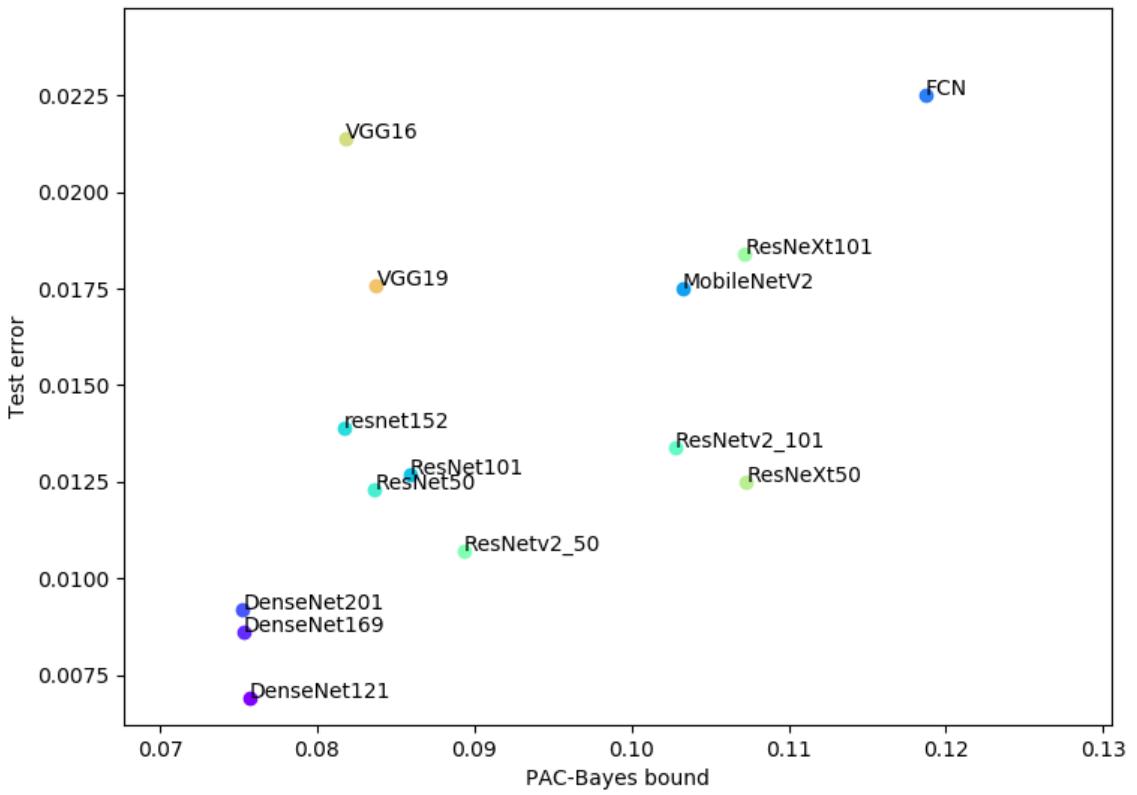


Figure F.18: PAC-Bayes bound versus test error for different models trained on a sample from MNIST of size 15k, with batch size 256. CNNs are removed for clarity as they often have relatively extreme values of test error and/or bound.

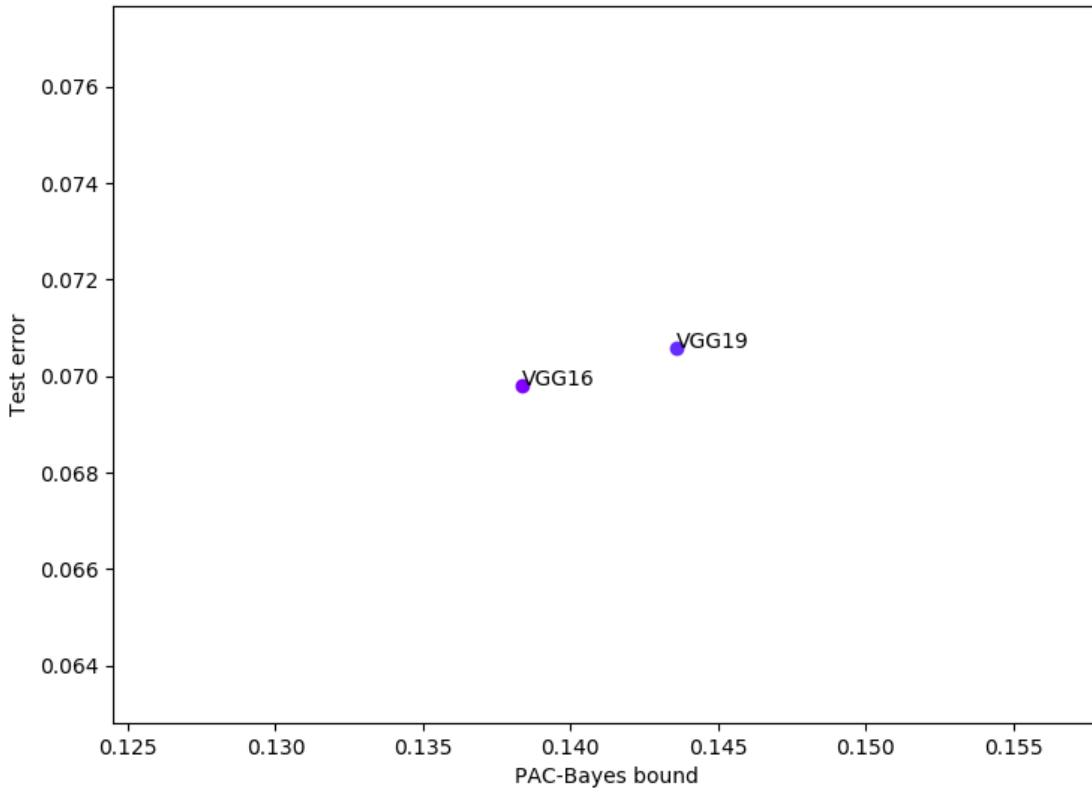


Figure F.19: PAC-Bayes bound versus test error for different models trained on a sample from Fashion-MNIST of size 15k, with batch size 256. CNNs are removed for clarity as they often have relatively extreme values of test error and/or bound.

F.7 Error versus bound for batch size 32, including CNN and FCN

In this section, we show the same experiments plotting the bound versus test error, as in Section 5.0.3, but including the CNN and FCN (which are often outliers), for completeness.

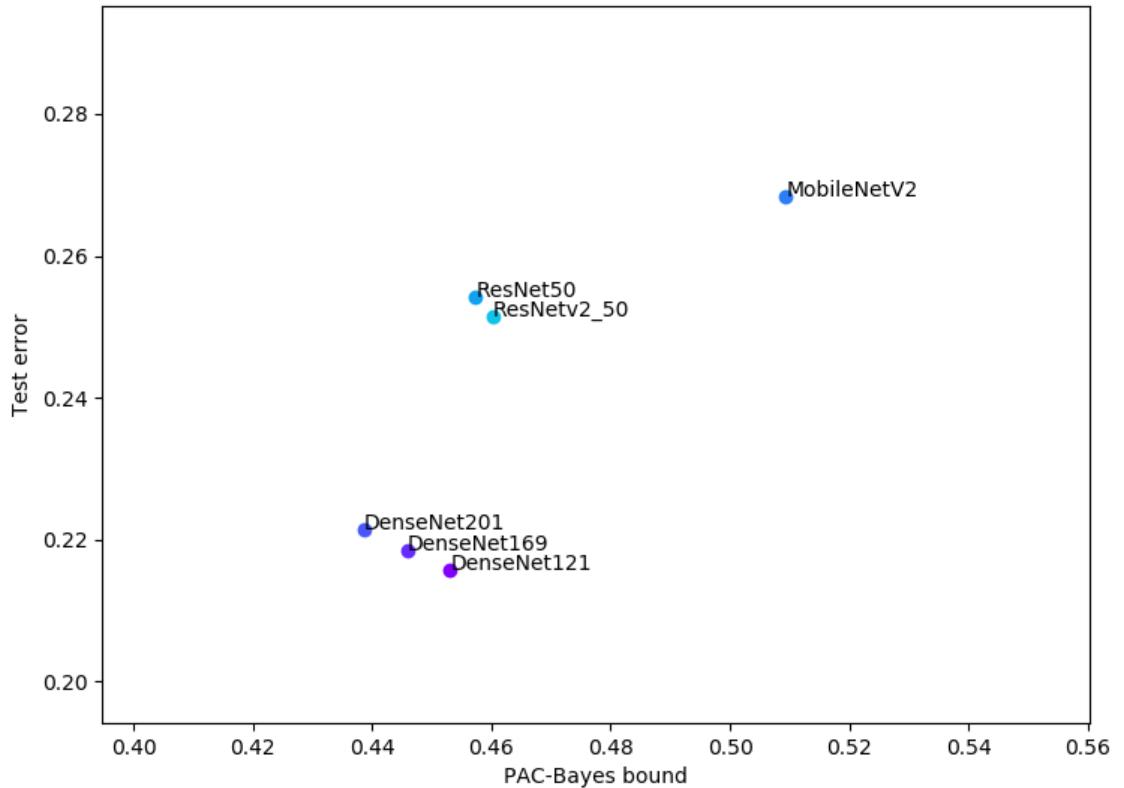


Figure F.20: PAC-Bayes bound versus test error for different models trained on a sample from CIFAR of size 15k, with batch size 32.

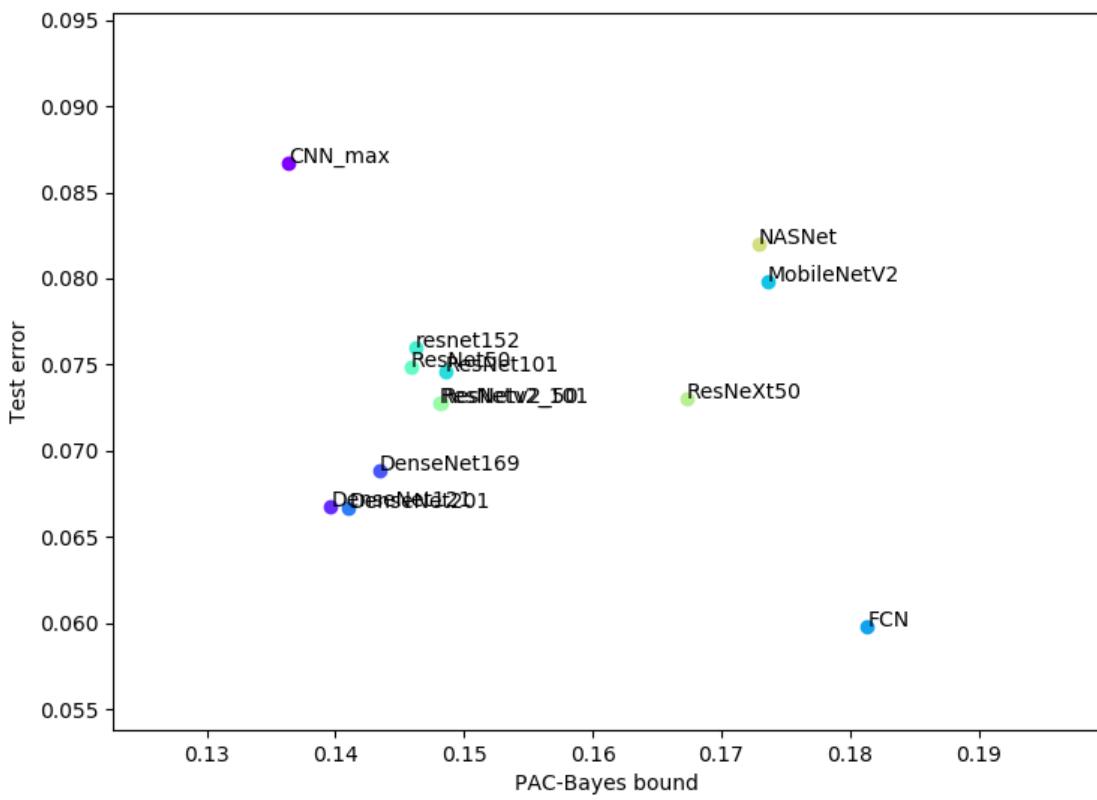


Figure F.21: PAC-Bayes bound versus test error for different models trained on a sample from EMNIST of size 15k, with batch size 32.

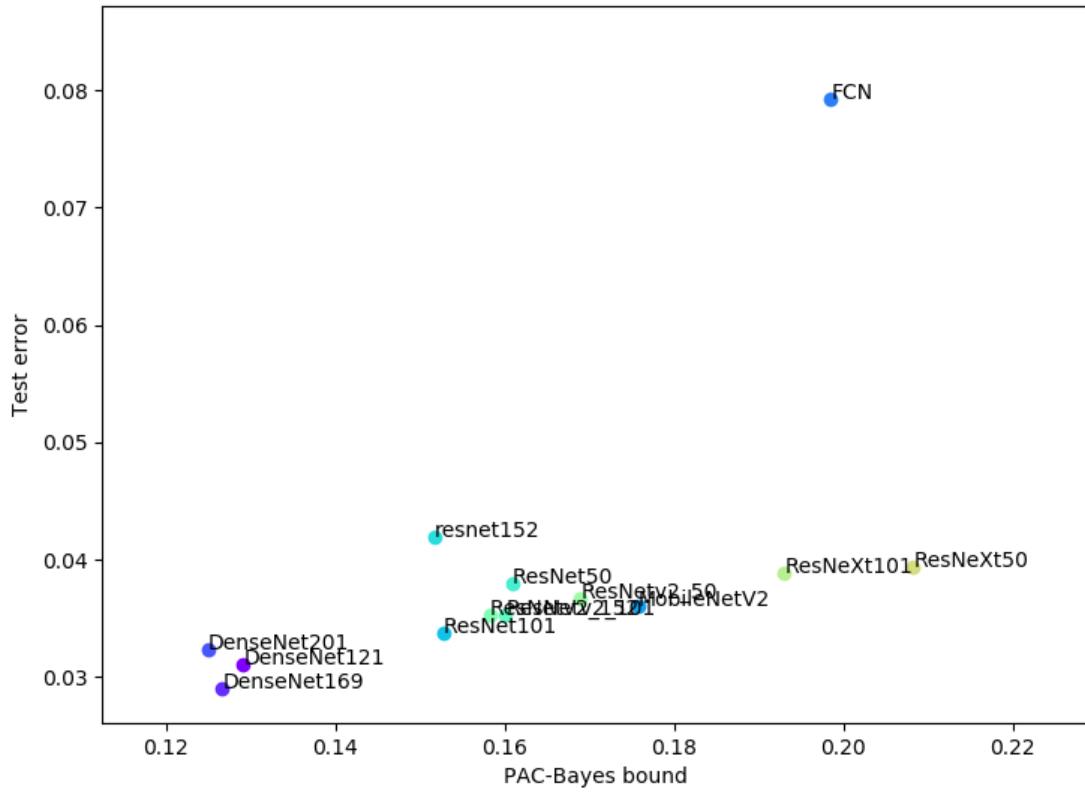


Figure F.22: PAC-Bayes bound versus test error for different models trained on a sample from KMNIST of size 15k, with batch size 32.

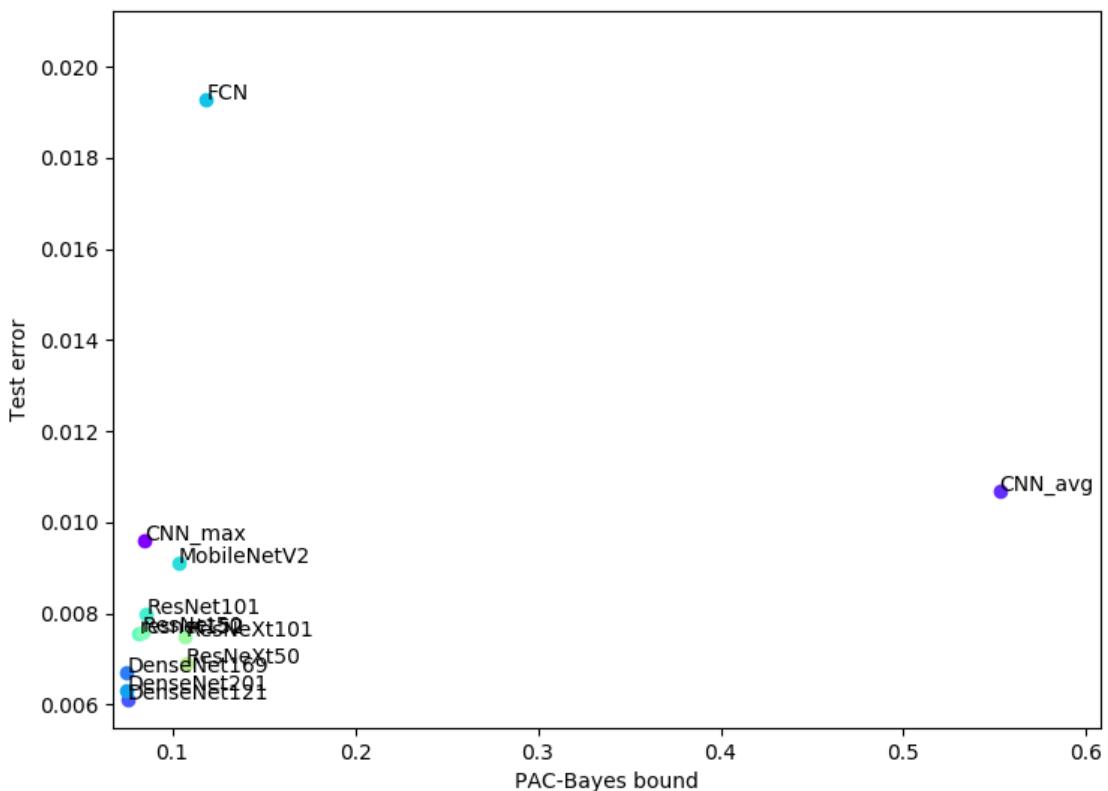


Figure F.23: PAC-Bayes bound versus test error for different models trained on a sample from MNIST of size 15k, with batch size 32.

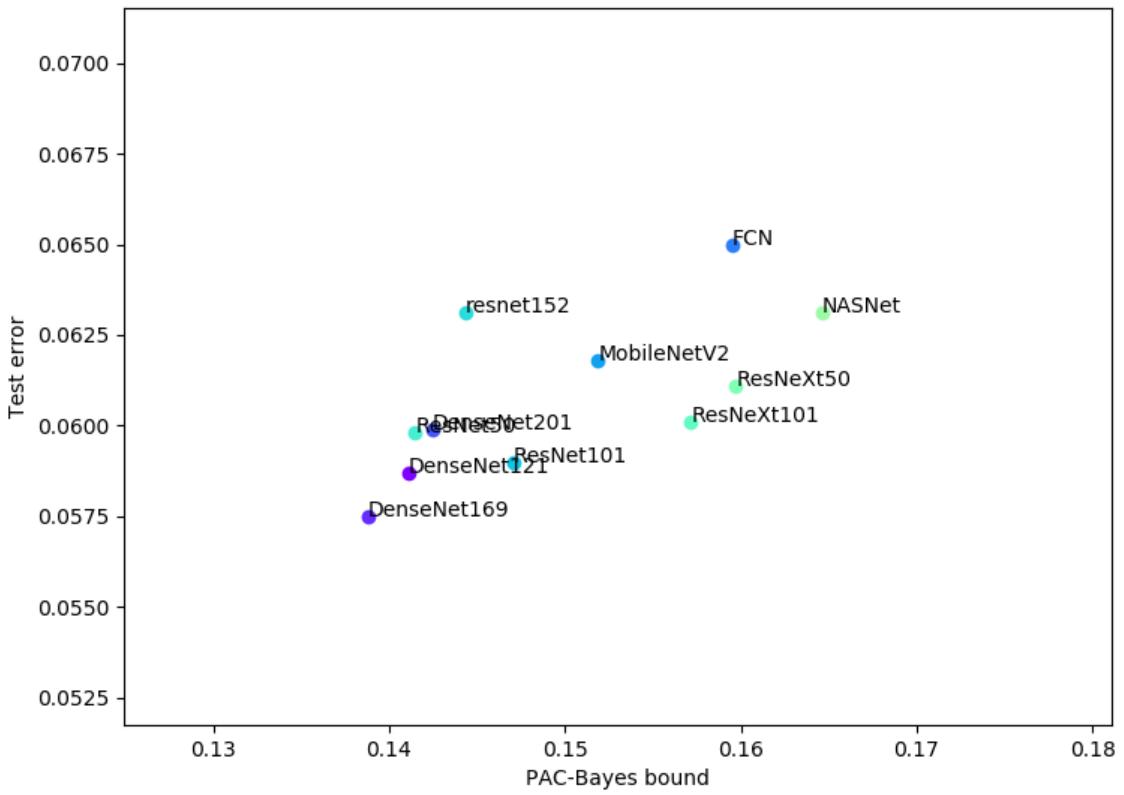


Figure F.24: PAC-Bayes bound versus test error for different models trained on a sample from Fashion-MNIST of size 15k, with batch size 32.

F.8 Error versus bound for batch size 256, including CNN

In this section, we show the same experiments plotting the bound versus test error, as in Section 5.0.3, but including the CNN (which is often an outlier), for completeness.

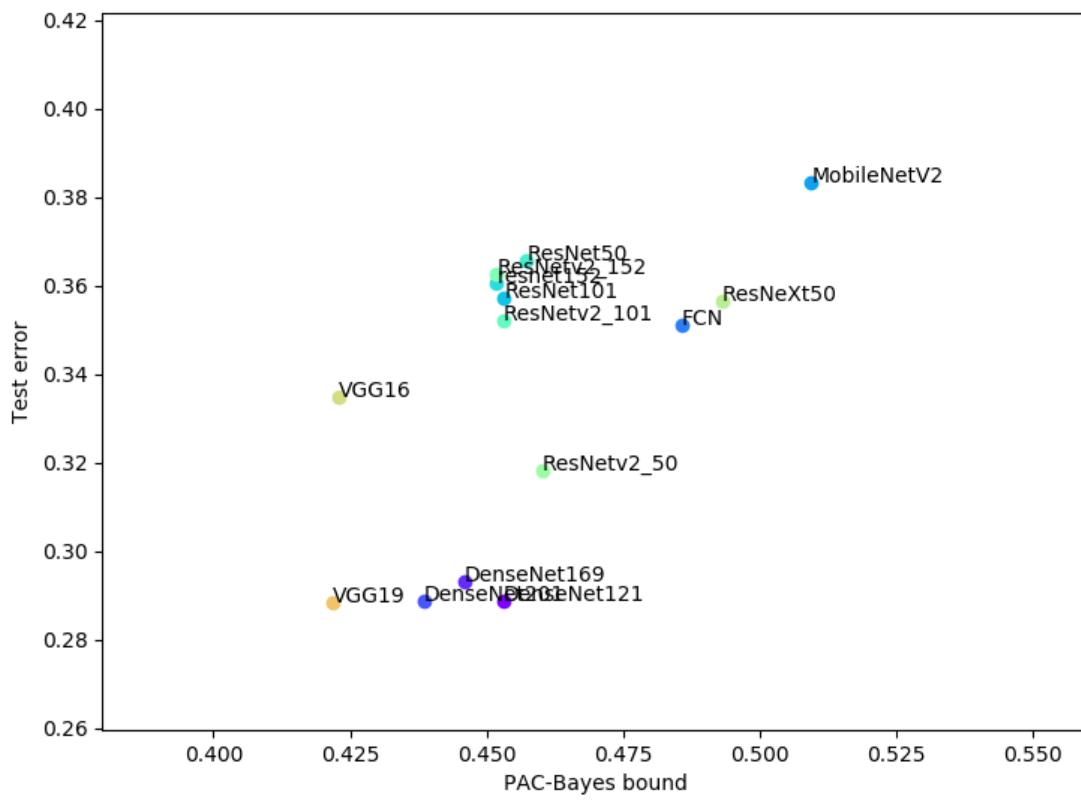


Figure F.25: PAC-Bayes bound versus test error for different models trained on a sample from CIFAR of size 15k, with batch size 256.

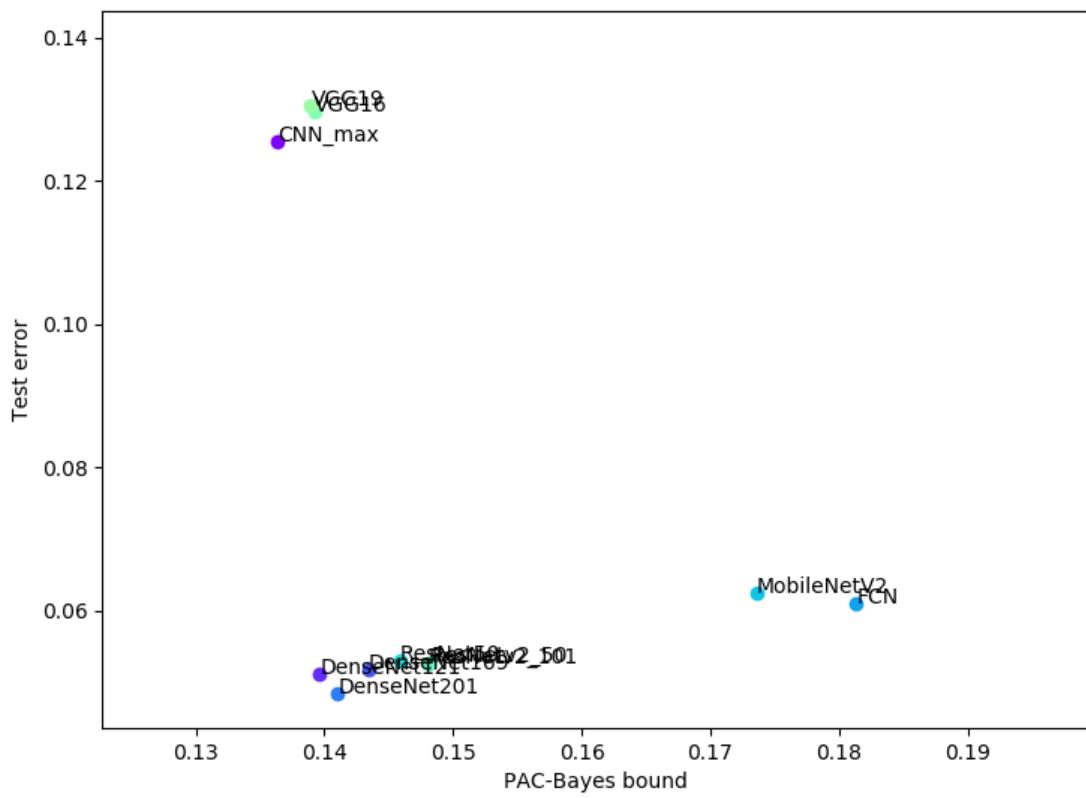


Figure F.26: PAC-Bayes bound versus test error for different models trained on a sample from EMNIST of size 15k, with batch size 256.

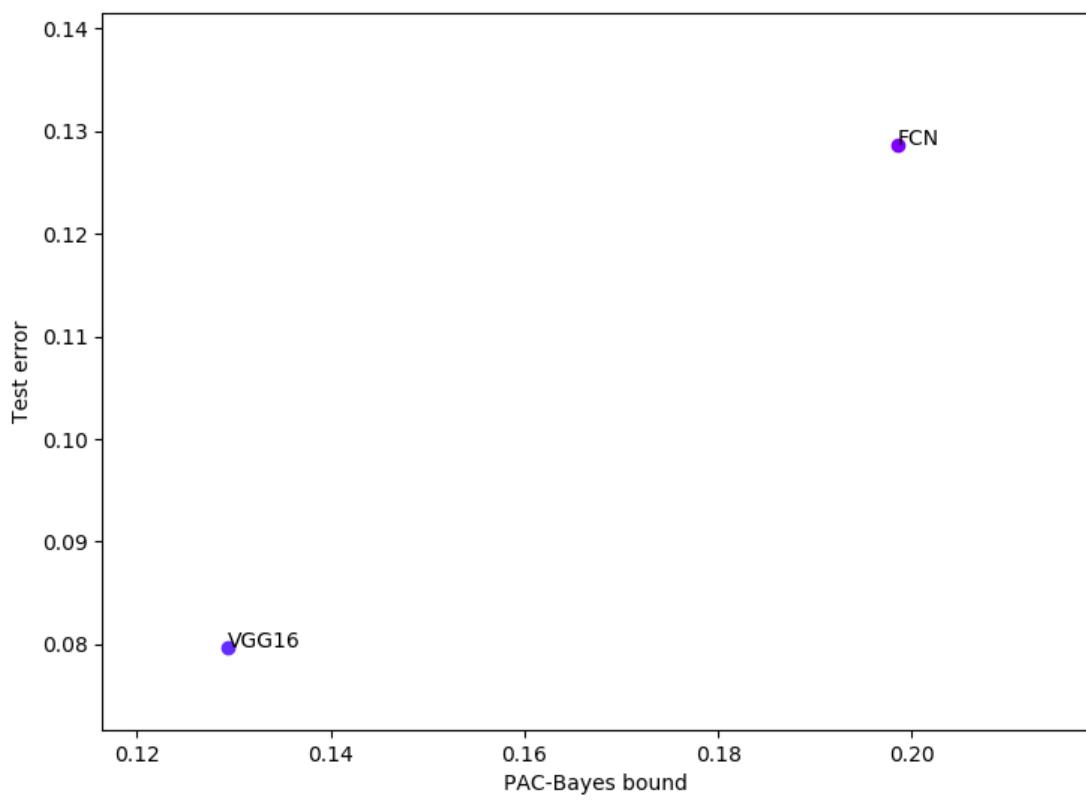


Figure F.27: PAC-Bayes bound versus test error for different models trained on a sample from KMNIST of size 15k, with batch size 256.

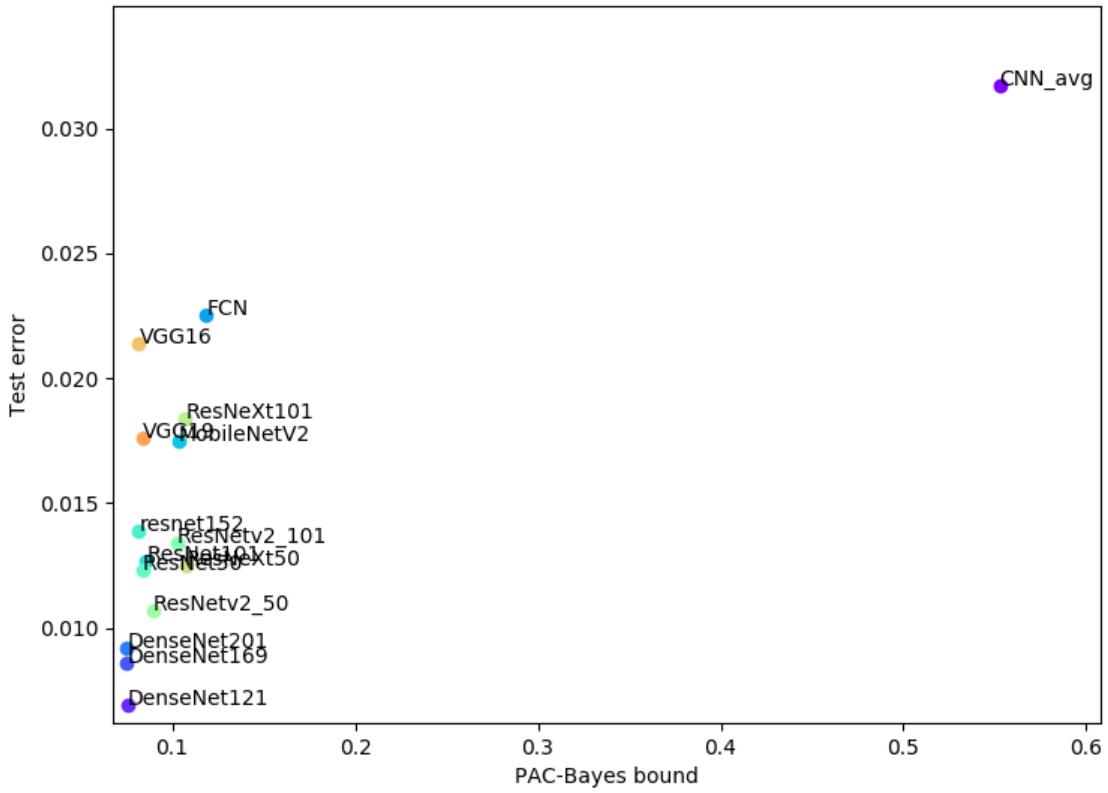


Figure F.28: PAC-Bayes bound versus test error for different models trained on a sample from MNIST of size 15k, with batch size 256.

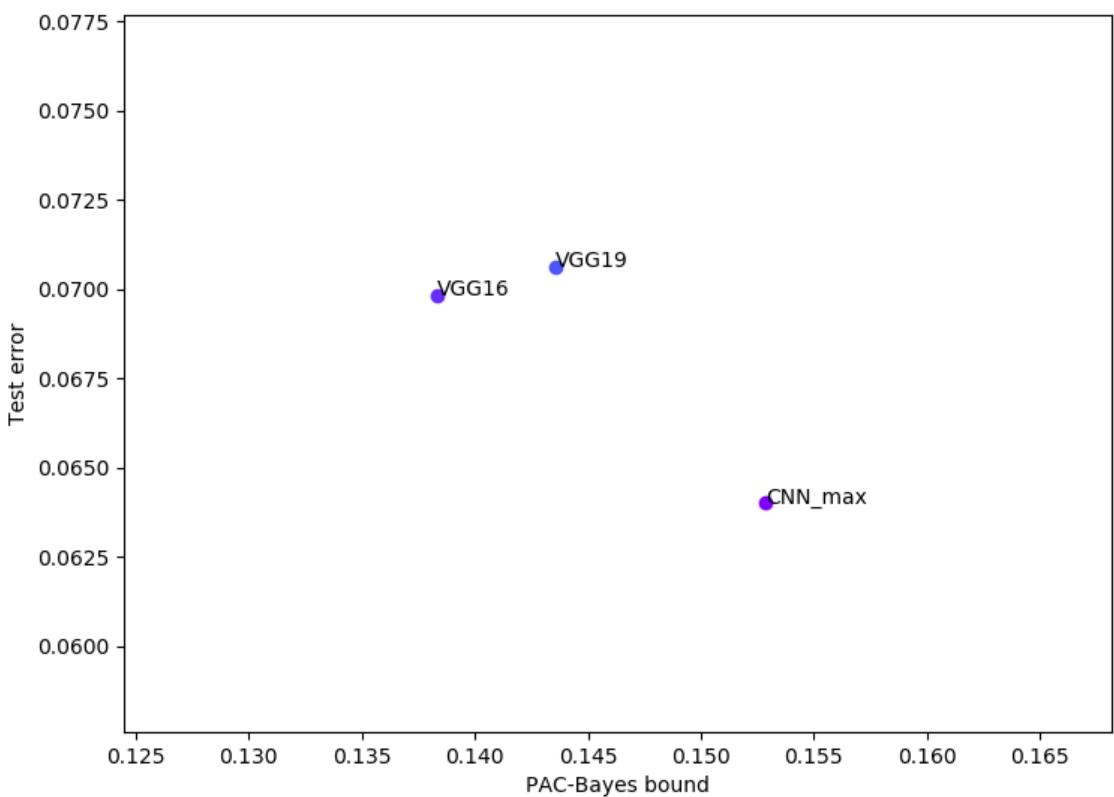


Figure F.29: PAC-Bayes bound versus test error for different models trained on a sample from Fashion-MNIST of size 15k, with batch size 256.

Appendix G

Further results on comparing SGD and Bayesian inference

In this appendix we present further results from the experiments in Chapter 6.

G.1 Further results comparing $P_{\text{OPT}}(f|S)$ to $P_{\text{B}}(f|S)$.

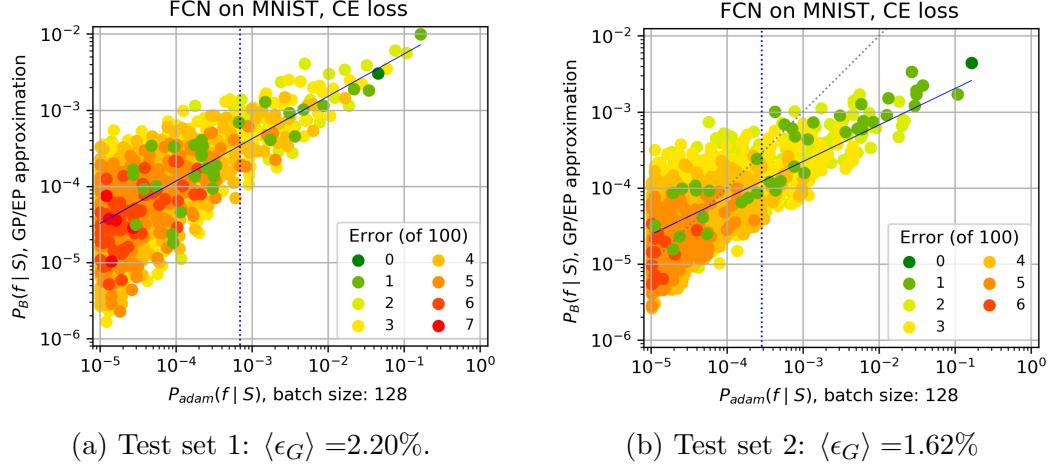


Figure G.1: Comparing $P_{\text{B}}(f|S)$ to $P_{\text{Adam}}(f|S)$ for an FCN on MNIST with CE loss for two different test sets [We use training/test set size 10,000/100 and batch size=128. Vertical dotted blue lines denote 90% probability boundary; solid blue lines are fit to guide the eye; dashed grey line is $x = y$.] (a) is the test set used throughout the paper and (b) is another test set (disjoint from both the first test set and the training set), chosen at random from MNIST. To first order the two look very similar, but to second order small differences can be seen, not just in the function probabilities, but also in the slope of $P_{\text{B}}(f|S)$ v.s. $P_{\text{Adam}}(f|S)$, which may be affected by the EP approximation used here. No normalisation was applied to this figure so it depicts the raw EP approximation (see Section B.3.1).

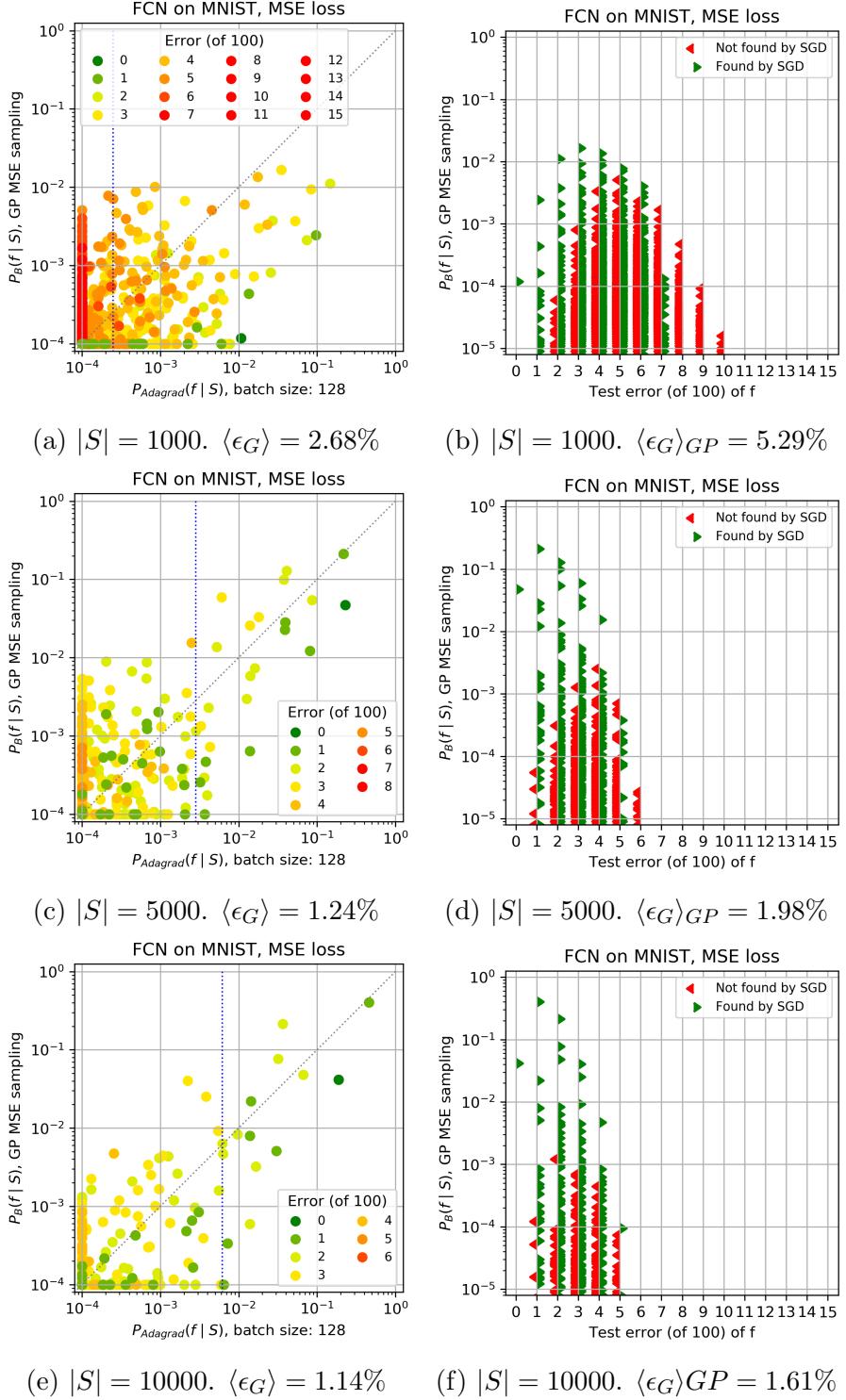


Figure G.2: $P_B(f|S)$ v.s. $P_{\text{Adagrad}}(f|S)$ for an FCN on MNIST with MSE loss, for different training set sizes. Further results for Figure G.11. [Test set size $|E| = 100$ and batch size=128. Vertical dotted blue lines denote 90% probability boundary; solid blue lines are fit to guide the eye; dashed grey line is $x = y$.] (a) and (b) show 1000 training examples, and (c) and (d) show 5000 training examples, and (e) and (f) show 10000. As expected, more training examples reduce the generalisation error and increase the correlation between $P_{\text{Adagrad}}(f|S)$ and $P_B(f|S)$. For all the larger training sets, the $\sum P_B(f|S) \approx 1$, but for $|S| = 1000$ functions found by GP sampling²⁵⁵ only make up about 40% of the probability of all functions found by Adagrad.

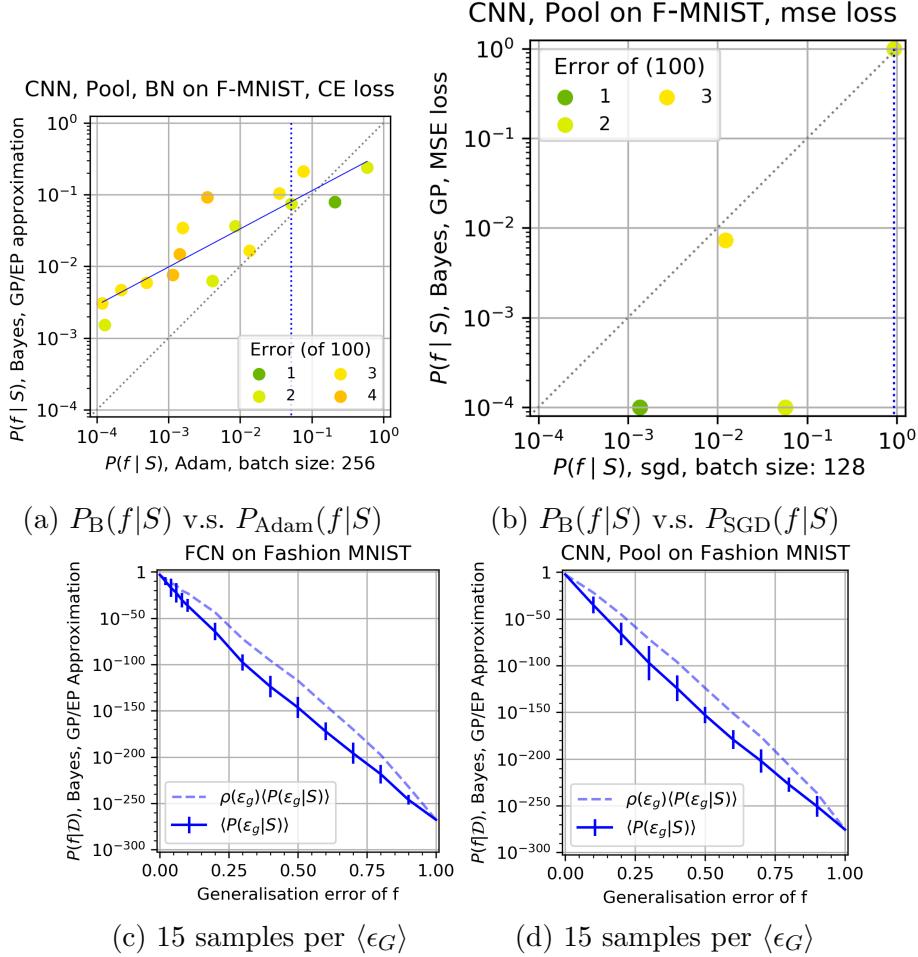


Figure G.3: Comparing the Bayesian prediction $P_B(f|S)$ to $P_{\text{Adam}}(f|S)$ for an FCN and CNNs on Fashion-MNIST. Further results for Figure 6.2. [We use train/test set size of 10,000/100; vertical dotted blue lines denote 90% probability boundary; solid blue lines are fit to guide the eye; dashed grey line is $x = y$.] (a) CNN with max-pooling and batch normalisation on Fashion-MNIST; $\langle \epsilon_G \rangle = 2.11\%$ for Adam with CE loss. Note that the GP kernel used is the same as in Figure 6.2c, so with pooling but without batch normalisation. The effect of batch normalisation is relatively small on this system. (b) CNN with max-pooling on Fashion-MNIST and MSE loss. $\langle \epsilon_G \rangle = 2.01$ for SGD and $\langle \epsilon_G \rangle = 2.00$ for the GP MSE sampling. 2000 samples for SGD, 10^5 samples from the GP. The same function is found over 99% of the time by both the GP and SGD. (c) and (d) show an average of $P_B(f|S)$ versus error for an FCN and CNN with max pooling respectively.

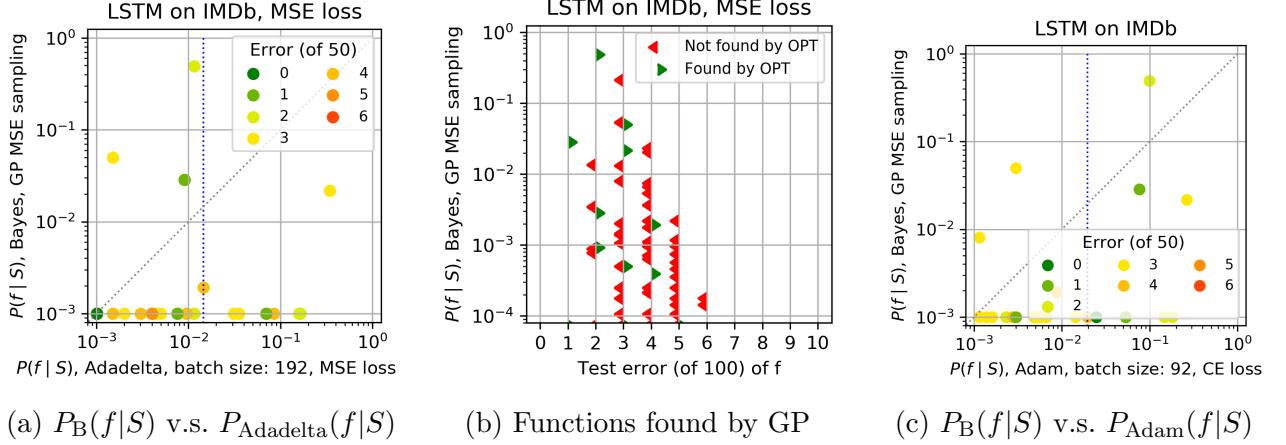


Figure G.4: **Comparing $P_B(f|S)$ to $P_{\text{OPT}}(f|S)$ for a LSTM on the IMDb movie review dataset. Further results for Figure 6.4.** [We use training/test set size 45,000/50 and batch size=192. Vertical dotted blue lines denote 90% probability boundary; solid blue lines are fit to guide the eye; dashed grey line is $x = y$.]

(a) $P_B(f|S)$ v.s. $P_{\text{Adadelta}}(f|S)$ for MSE loss. $n = 2200$ for the optimiser and $n = 2.6 * 10^4$ for the GP. $\langle \epsilon_G \rangle = 5.22\%$ and $\langle \epsilon_G \rangle_{GP} = 5.04\%$

(b) Probability of all functions found by NNGP compared to those also found by the optimiser. Green points are the set jointly found functions F . Red denotes functions found only by the GP, $\sum_{f \in F} P_B(f|S) = 54.1\%$ and $\sum_{f \in F} P_{\text{OPT}}(f|S) = 77.0\%$.

(c) Compares the $P_{\text{Adam}}(f|S)$ with CE loss from Figure 6.4a, to the sampled $P_B(f|S)$ using MSE loss for $n = 10^4$ samples.

While these results are for very limited sample numbers, they provide evidence that for the LSTM, $P_B(f|S)$ has values on the same order of magnitude as $P_{\text{SGD}}(f|S)$. The low values we find for the raw EP approximation estimates of $P_B(f|S)$ are likely to be due to errors in the EP absolute values. The fact that we still see correlations for the CE-trained LSTM with the renormalised EP approximations for $P_B(f|S)$ suggests that the EP still does reasonably on relative errors. CE has the advantage that it is much faster to use than MSE.

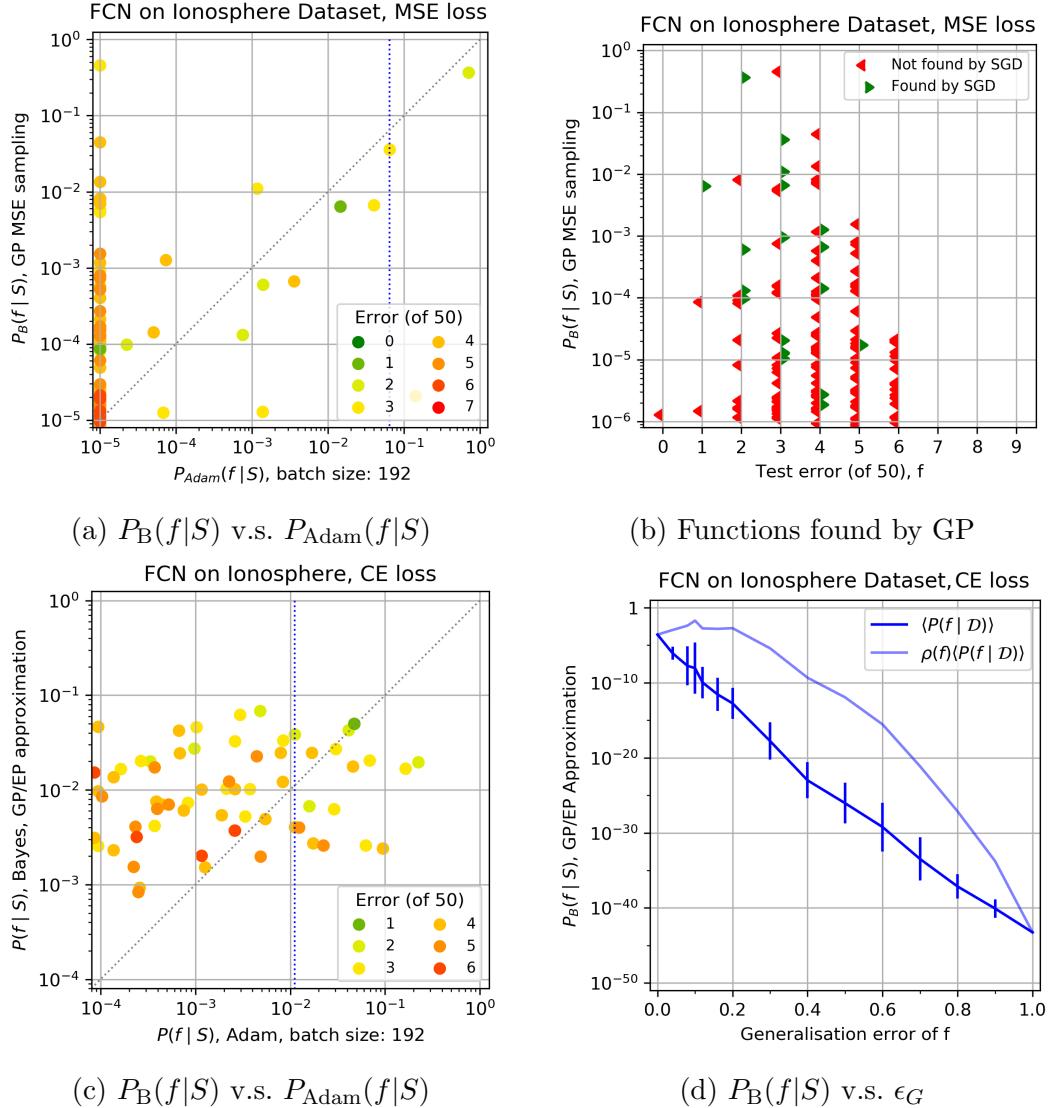


Figure G.5: **Comparing $P_B(f|S)$ to $P_{Adam}(f|S)$ for an FCN on the Ionosphere dataset with CE loss. Further results for Figure 6.4.** [We use training/test set size 301/50 and batch size=192. Vertical dotted blue lines denote 90% probability boundary; solid blue lines are fit to guide the eye; dashed grey line is $x = y$.]

- (a) $P_B(f|S)$ v.s. $P_{Adam}(f|S)$ for MSE (same as in Figure 6.4), put here ease of comparison).
- (b) Probability of all functions found by NNGP compared to those also found by the optimiser. Green points are the set jointly found functions F . $\sum_{f \in F} P_B(f|S) = 43.1\%$ and $\sum_{f \in F} P_{Adam}(f|S) = 99.8\%$ (in other words nearly all functions found by Adam are also found by the GP, but the GP also finds functions that Adam doesn't for this level of sampling). For the optimiser, $\langle \epsilon_G \rangle = 4.59\%$ and for the GP MSE sampling, $\langle \epsilon_G \rangle_{GP} = 5.41\%$.
- (c) $P_B(f|S)$ v.s. $P_{Adam}(f|S)$ for CE. $\langle \epsilon_G \rangle = 5.88\%$.
- (d) $\langle P_B(f|S) \rangle$ versus $\langle \epsilon_G \rangle$ with CE, 20 samples per ϵ_G . The bias towards low error functions is less strong than what is found for MNIST or Fashion-MNIST.

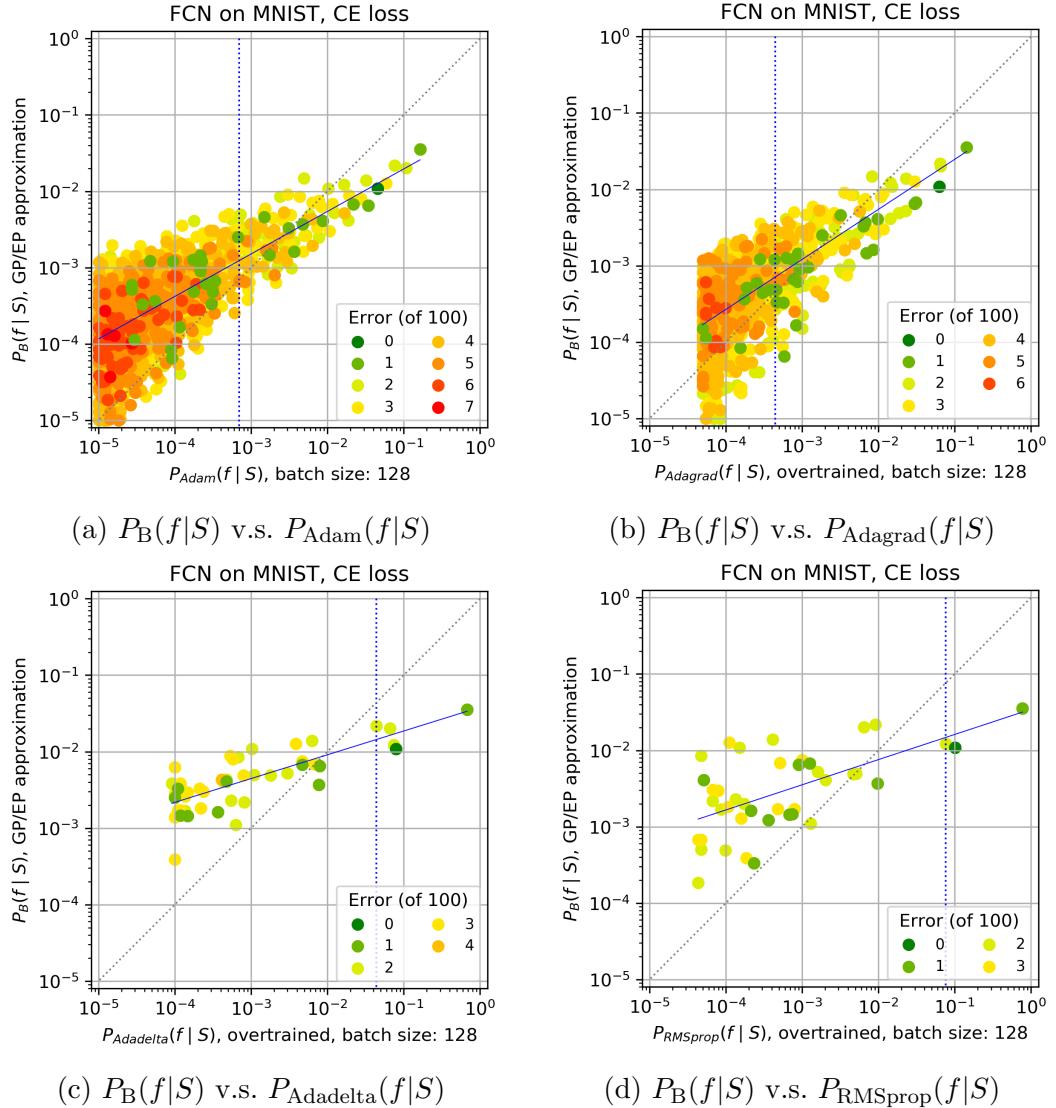


Figure G.6: Comparing $P_B(f|S)$ to $P_{\text{OPT}}(f|S)$ for an FCN on MNIST with CE loss for different optimisers. Further results for Figure G.10. [We use training/test set size 10,000/100; batch size=128. Vertical dotted blue lines denote 90% probability boundary; solid blue lines are fit to guide the eye; dashed grey line is $x = y$.]

- (a) Adam, no overtraining, $\langle \epsilon_G \rangle = 2.20\%$. Sample size: $n = 10^6$.
- (b) Adagrad with overtraining, $\langle \epsilon_G \rangle = 2.19\%$. Sample size: $n = 2.1 \times 10^5$.
- (c) Adadelta with overtraining, $\langle \epsilon_G \rangle = 1.17\%$. Sample size: $n = 10^5$.
- (d) RMSprop with overtraining, $\langle \epsilon_G \rangle = 1.01\%$. Sample size: $n = 2.5 \times 10^5$.

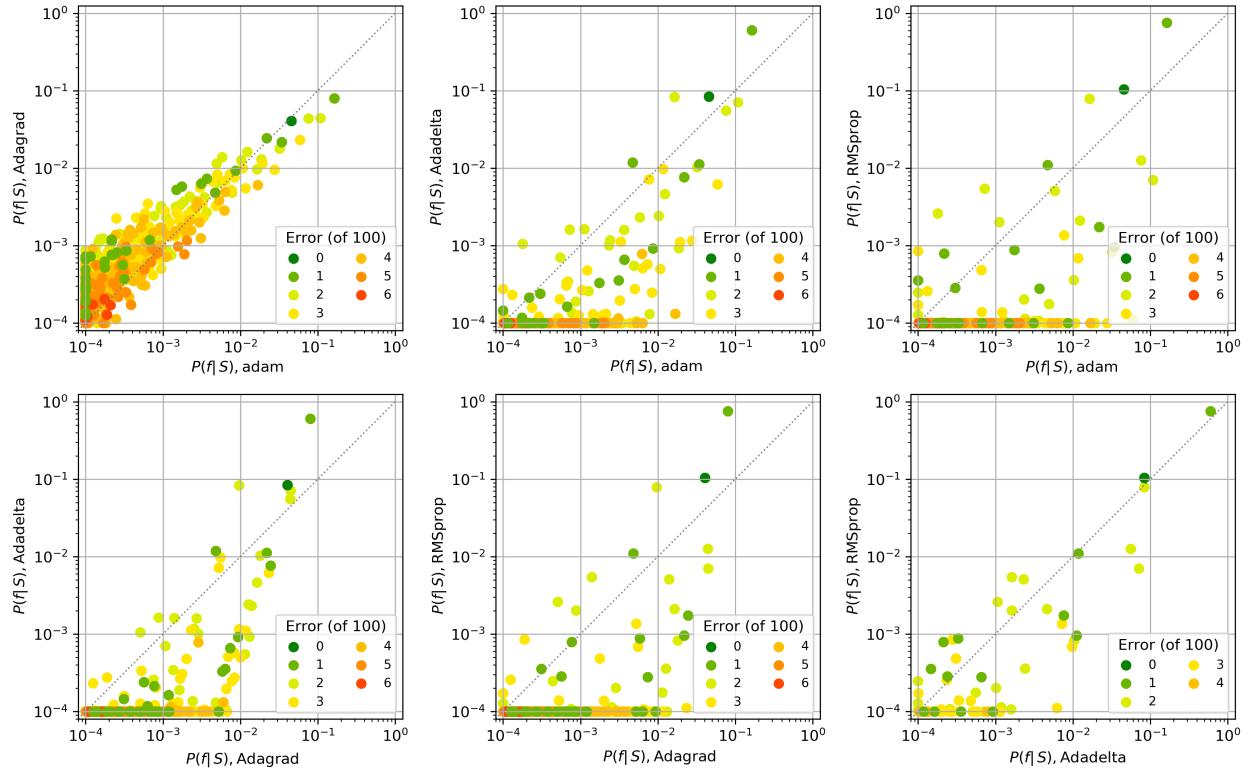
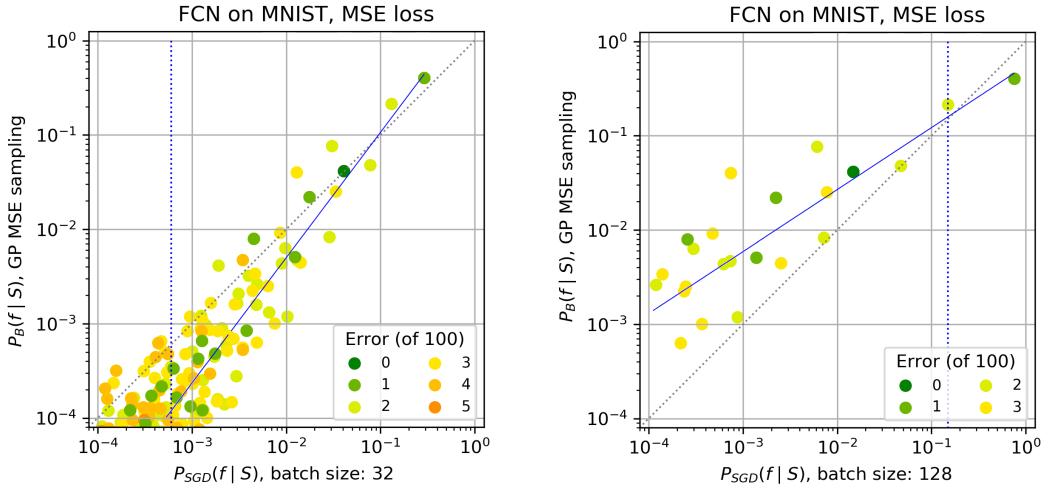


Figure G.7: Comparing $P_{\text{OPT}}(f|S)$ with $P_{\text{OPT}}(\hat{f}|S)$ for different optimisers. Further results for Figure G.10. [We use the FCN architecture on MNIST, CE loss and a batch size of 128, training/test set size = 10,000/100, $y = x$ is denoted by a dashed line]

Adagrad and Adam correlate very well as do Adadelta and RMSprop for these hyperparameters. The other combinations do not correlate as well, suggesting that they sample the loss-function differently from one another. These plots do not rely on the GP or GP/EP approximation. We believe that this sort of experiment may prove useful for understanding differences in the behaviour of the optimisers.



(a) $\langle \epsilon_G \rangle = 1.88\%$, 0 Test error opt/gp:

(b) $\langle \epsilon_G \rangle = 1.22\%$, 0 Test error opt/gp:

Figure G.8: $P_B(f|S)$ v.s. $P_{Adam}(f|S)$ for an FCN on MNIST with the Adam optimiser and MSE loss with different batch sizes. Further results for Section 6.5.1. Batch sizes: (a) 32, and (b) 128. For this MSE loss function, we observe that increasing the batch size leads to better generalisation, and also to a shallower best fit because the highest probability functions are sampled with enhanced probability by SGD. This trend with batch size is the opposite of what was observed for CE loss in Figure 6.5.

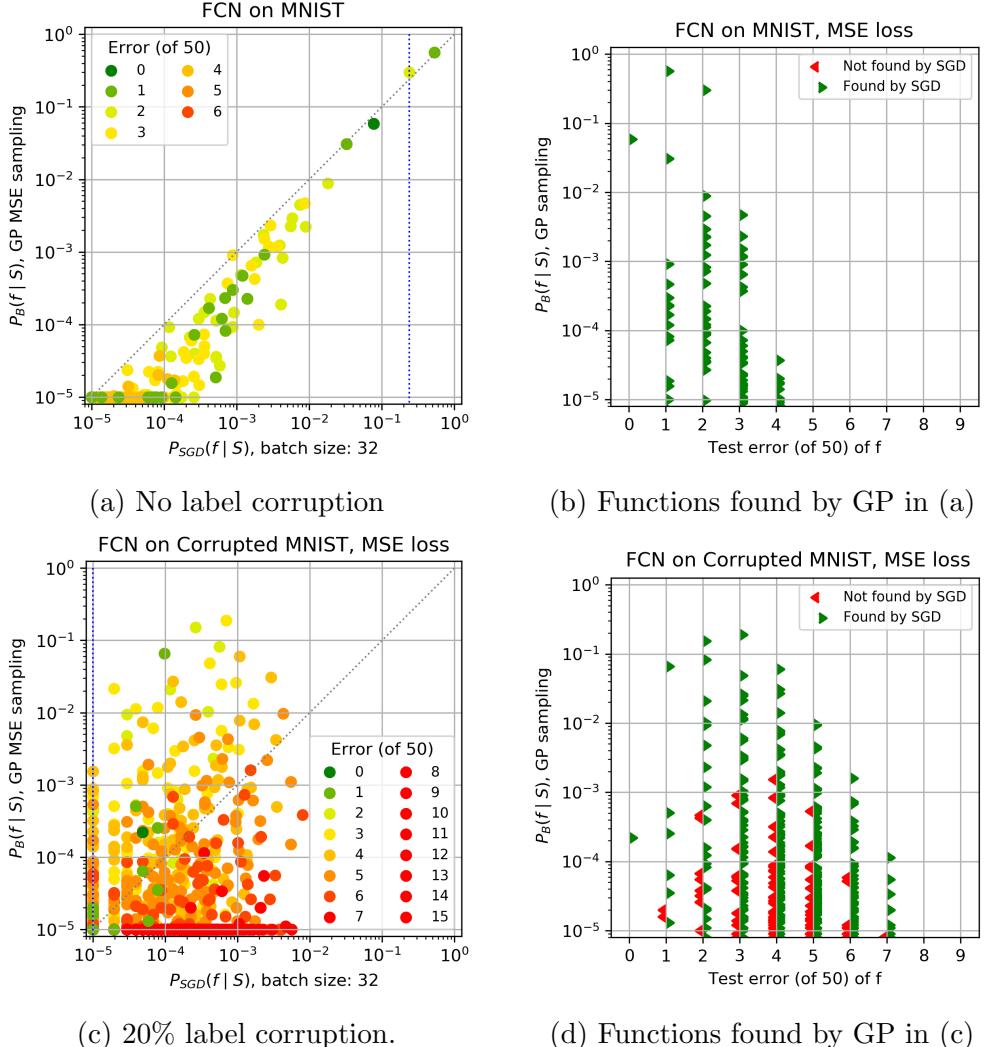


Figure G.9: **Comparing $P_{SGD}(f|S)$ to $P_B(f|S)$ for an FCN on MNIST with label corruption and MSE loss. Further results for Figure 6.6.** [Training/test set size 10,000/50; batch size=128; vertical dotted blue lines denote 90% probability boundary; solid blue lines are fit to guide the eye; dashed grey line is $x = y$; points on the axes are found by one technique only.]

(a) $P_B(f|S)$ v.s. $P_{SGD}(f|S)$ for no corruption, $n = 10^6$, $\langle \epsilon_G \rangle = 2.64\%$ for SGD and $\langle \epsilon_G \rangle_{GP} = 3.22\%$ for the GP. (Note this relative error is larger than for the $|E| = 100$ test set because it includes a hard to classify image. Such fluctuations are expected for small test sets.)

(b) Green dots denote functions in the set F , which are found by both SGD and the GP sampling. Red dots are found by GP, but not by SGD. On this scale all GP functions are found by SGD.

(c) $P_B(f|S)$ v.s. $P_{SGD}(f|S)$ for 20% corruption (as Figure 6.6(c), but included for ease of comparison). ($\langle \epsilon_G \rangle = 13.4\%$ for SGD and $\langle \epsilon_G \rangle_{GP} = 5.80\%$ for the GP). Here we included functions with frequency < 10 . $n = 10^5$

(d) In contrast to (b), a considerable number of low probability functions are not found by SGD. Here $\sum_{f \in F} P_B(f|S) = 99.3\%$, and $\sum_{f \in F} P_{SGD}(f|S) = 24.3\%$, indicating that while SGD finds almost all functions with high $P_B(f|S)$, it also finds many functions with low $P_B(f|S)$. Comparing the 0% and 20% label corruption shows that the weaker bias in the latter leads to less strong correlation between $P_B(f|S)$ and $P_{SGD}(f|S)$.

G.1.1 Changing optimisers

Much research effort has gone into adaptations of SGD. One goal is to achieve more efficient optimisation, but another is to achieve better generalisation. Figure G.10 illustrates the effect of changing the optimiser on the correlation between $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$. (See also Figures 6.1f, 6.5b and G.6 to complete the set of optimisers with and without overtraining). To first order this figure shows that $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$ are remarkably closely correlated for all these optimisers, even taking into account that the EP approximation introduces errors, and likely leads to a slightly too small slope in $P_B(f|S)$ v.s. $P_{\text{OPT}}(f|S)$.

What is perhaps more interesting here are second-order effects, since $P_B(f|S)$ is identical in each plot. For example, RMSprop has the best generalisation performance, which is reflected in a stronger inductive bias towards a few key low error functions. Note also the similarity of batch size 128 RMSprop to Adam with smaller batch size of 32. We emphasise that this performance here doesn't mean that RMSprop is in general superior to the other SGD variants for FCNs on MNIST. To investigate that question, we would need to study other test and training sets, and need to do further hyperparameter tuning [Choi et al., 2019].

We can also compare the effect of overtraining. In each case shown in Figures 6.1f, 6.5b and G.6, overtraining brings a modest improvement in generalisation error on this test set (Adam from $\epsilon_G = 2.2\%$ to $\epsilon_G = 1.74\%$ and Adagrad from $\epsilon_G = 2.63\%$ to $\epsilon_G = 2.19\%$), for example. From the graphs one can see a slight increase in the optimiser probability of the lowest error function with overtraining, but also a clear reduction in the scatter of the data for the whole range of probabilities, suggesting that for CE loss, on average, overtraining brings $P_{\text{OPT}}(f|S)$ closer to the Bayesian prediction. This behaviour can possibly be rationalised in that overtraining allows the optimiser to sample functions with probabilities closer to the steady-state average (see also Section 6.6).

Finally, in Figures G.7, G.10e and G.10f we directly compare the $P_{\text{OPT}}(f|S)$ to one another, in other words, without using $P_B(f|S)$. A number of clear trends are visible, for example, with batch size 128, Adam and Adagrad are very similar to one another, as are RMSprop and Adadelta. However, as can be seen in Figure G.10f, Adam with a smaller batch size of 32 is very similar to RMSprop with batch size of 128. What these correlation plots show is

that the behaviour of the different optimisers can depend on batch size in subtle ways that may not necessarily be picked up by the generalisation error. Further work (and significant computational resources) would be needed to completely compare these methods.

These examples show that studying the spectrum of function probabilities provides more fine-grained data than simply comparing generalisation error does. Future studies on problems such as optimiser choice or hyperparameter tuning could exploit this fuller set of information to increase understanding and to improve DNN performance.

G.1.2 Effects of training set size

It is also instructive to study the correlation between $P_B(f|S)$ and $P_{\text{OPT}}(f|S)$ for different training set sizes $|S|$. As can be seen clearly in Figure G.11, as the training set increases in size, the functions with zero training error are more strongly biased towards low generalisation error, as expected. Figures G.11c and G.11f also illustrate how the stronger bias with increasing $|S|$ means that the entropic factor $\rho(\epsilon_G)$ plays a smaller role. Thus, for larger training set size, but for the same amount of sampling n , fewer functions are found, but on average they have higher probability.

An important question in deep learning is: How does the error reduce with increasing the training set size? There is intriguing evidence that such “learning curves” follow a power law that depends on data complexity, and only weakly on the architecture [Hestness et al., 2017, Spigler et al., 2019, Rosenfeld et al., 2019, Kaplan et al., 2020]. Figure G.11 shows how the spectrum of function probabilities changes with increasing $|S|$. Investigations based on this more fine-grained picture may help improve our understanding of learning curves.

G.2 Bayesian inference by direct sampling on Boolean system.

In this section we consider a much simpler system, with a smaller input space, and thus a much smaller space of functions. This allows us to approximate Bayesian inference in the case of 0-1 likelihood (see Section B.3.1) much more accurately, via direct sampling. We use the same

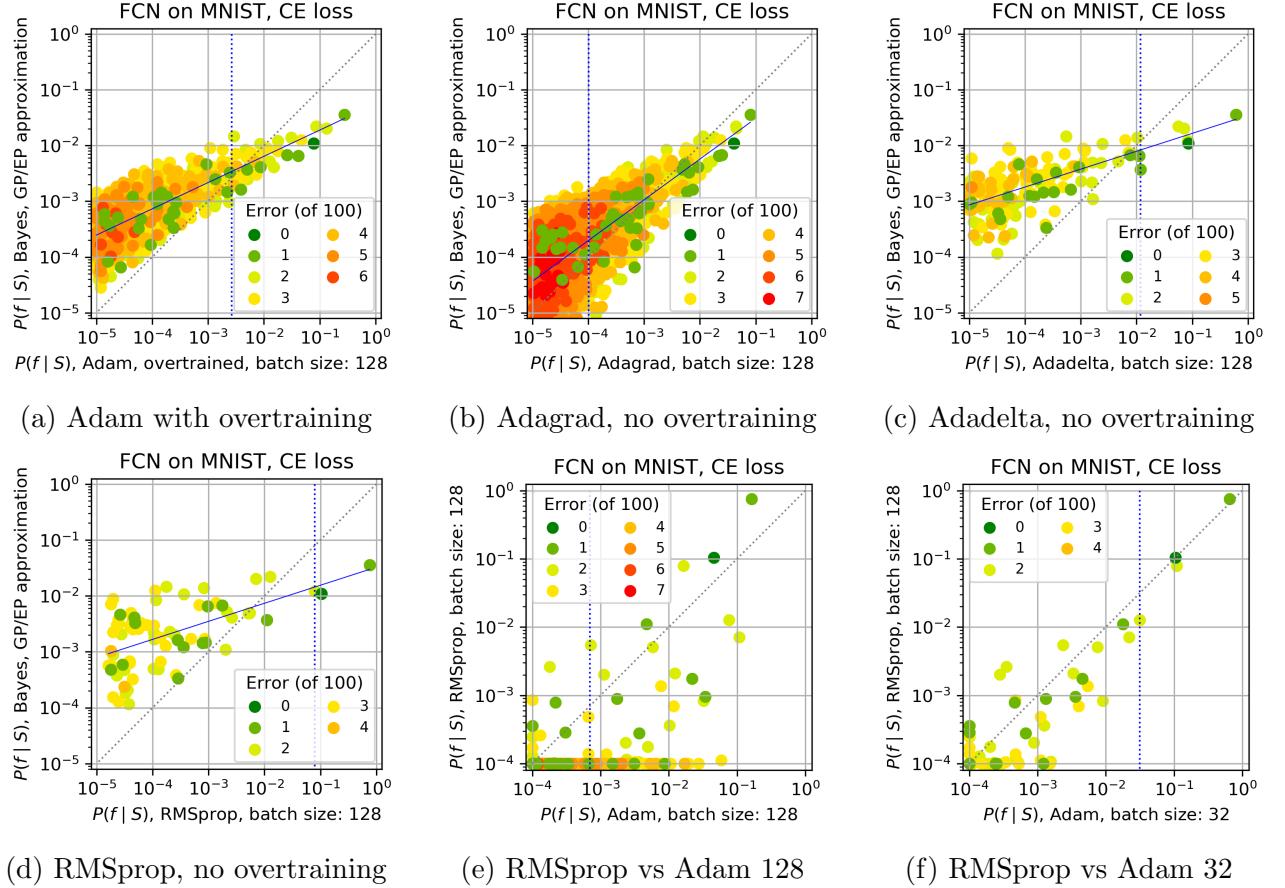


Figure G.10: Comparing $P_B(f|S)$ to $P_{OPT}(f|S)$ for an FCN on MNIST with CE loss for different optimisers. [We use training/test set size 10,000/100 and batch size=128. Vertical dotted blue lines denote 90% probability boundary; dashed grey line is $x = y$.]

(a) Adam with overtraining for 64 epochs. $\langle \epsilon_G \rangle = 1.73\%$.

(b) Adagrad, $\langle \epsilon_G \rangle = 2.63\%$.

(c) Adadelta, $\langle \epsilon_G \rangle = 1.23\%$.

(d) RMSprop, $\langle \epsilon_G \rangle = 1.02\%$.

(e) $P_{RMSprop}(f|S)$ v.s. $P_{Adam}(f|S)$ both with batch size 128.

(f) $P_{RMSprop}(f|S)$ with batch size 128 v.s. $P_{Adam}(f|S)$ with batch size 32.

For Adam without overtraining, see Figure 6.5b. For Adagrad with overtraining see Figure 6.1f. For others with overtraining see Figure G.6. Overtraining has a milder effect than changing the optimiser does. (e) and (f) show that there is a surprisingly close correspondence between RMSprop with batch size 128 and Adam with batch size 32. By contrast, for the same batch size of 128, Adam and Adagrad are similar to one another, as are Adadelta and RMSprop (see Figure G.7 for a direct comparison). The latter two have better generalisation performance, which is reflected in higher probabilities for the lowest error functions. See Figures G.6 and G.7 for further results.

In all these figures, while $P_{OPT}(f|S)$ is to first order determined by $P_B(f|S)$, there are noticeable second order differences.

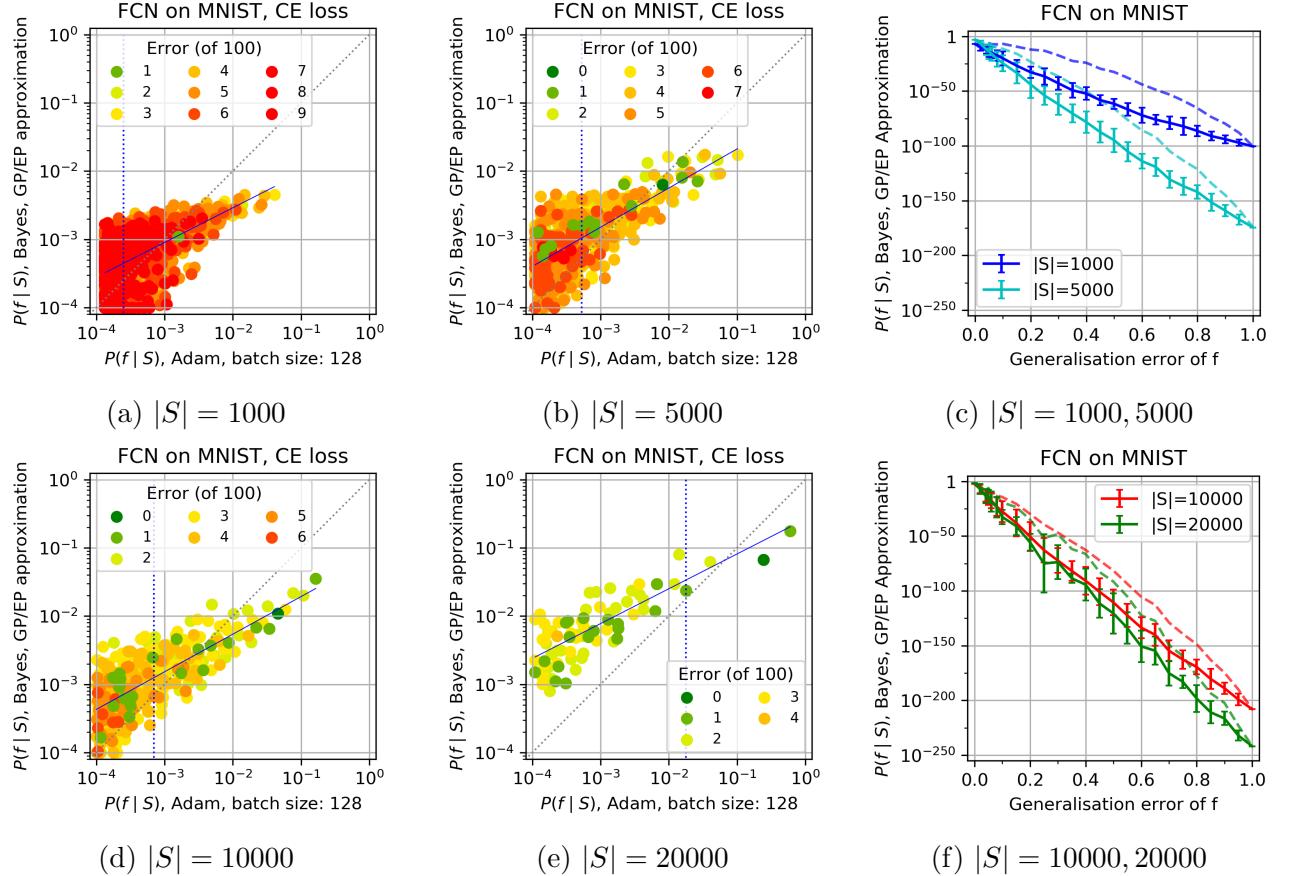


Figure G.11: **Comparing $P_B(f|S)$ to $P_{\text{Adam}}(f|S)$ for an FCN on MNIST with CE loss for different training set sizes.** [We use test set size of $|E| = 100$; vertical dotted blue lines denote 90% probability boundary; solid blue line is a guide to the eye, dashed grey line is $x = y$.]

- (a) 1000 training examples. $\langle \epsilon_G \rangle = 6.65\%$ for Adam.
- (b) 5000 training examples. $\langle \epsilon_G \rangle = 3.33\%$ for Adam.
- (c) $P_B(f|S)$ v.s. ϵ_G for 1000 and 5000 training examples.
- (d) 10000 training examples. $\langle \epsilon_G \rangle = 2.20\%$ for Adam.
- (e) 20000 training examples. $\langle \epsilon_G \rangle = 0.89\%$ for Adam.
- (f) $P_B(f|S)$ v.s. ϵ_G for 5000 and 10,000 training examples.

A trend of increasing bias towards lower error functions with increasing training set size can be clearly observed. See Figure G.2 for related results with MSE loss.

DNN architecture and synthetic data studied in [Valle-Pérez et al., 2018]. It consists of a two layer neural network with 7 Boolean inputs, two hidden layers of 40 ReLU-activated neurons, and a single Boolean output. The space of functions is thus the space of $2^{128} \approx 3.4 \times 10^{38}$ Boolean functions of 7 inputs.

We perform ‘approximate Bayesian inference’ (ABI) by sampling the parameters of the neural network i.i.d. from a Gaussian distribution with distribution parameters¹ $\sigma_w = \sigma_b = 1.0$, evaluating the neural network on the training set, and saving the samples for which the neural network achieves 100% training accuracy. Each of these samples corresponds to a function, **sampled from the *exact* Bayesian posterior**. We estimate the posterior probabilities by the empirical frequencies of individual functions (defined on all $2^7 = 128$ inputs), after sampling the parameters 10^{10} times. We used a random but fixed training set consisting of 32 out of the 128 inputs for a given target function. Target functions were chosen among functions that appeared with reasonably high frequency, in a large sample obtained by randomly sampling the weights of the neural network, so as to ensure ABI would give enough samples. They were chosen to have a range of values of Lempel-Ziv complexity. See [Valle-Pérez et al., 2018] for the definition of Lempel-Ziv complexity of Boolean functions used here.

Representative results are shown in Figures G.12,G.13,G.14 (results for the other 4 functions we tested look qualitatively similar). We empirically found that the ABI probabilities correlated and are of a similar order of magnitude to the SGD probabilities over the whole range of Boolean functions tried. SGD is consistently more biased towards the most likely functions for CE loss, although it only increased their probability by about a factor of two (a small amount relative to the whole range of probabilities, but because for this system this is the dominant function, this secondary effect can still have a significant effect on the average generalisation error). We also performed sampling using the EP approximation to the posterior with 0-1 loss, and the exact posterior using MSE loss to directly see the effects coming from the EP approximation. We found that for some functions (the simplest ones) GP/EP gave probabilities which were close to those found by ABI. However, for more complex functions² GP/EP highly

¹Remember that, following standard convention, the actual weight variance is dividing by the number of input neurons

²These are still rather simple w.r.t. the full range of possibilities, necessary to ensure that ABI sampling is feasible.

underestimates the probabilities. In fact, for the most complex functions we studied, GP/EP didn't find a single function more than once in our sampling. This is in contrast to the GP/MSE sampling probabilities, which shows reasonable correlation with the ABI probabilities, as well as with the probabilities of SGD trained with MSE loss.

These results support the main hypothesis which we proposed at the beginning of the chapter that the Bayesian posterior probabilities correlate with the SGD probabilities. They also suggest that the EP approximation can sometimes heavily underestimate the probabilities. This agrees with the conjecture that the EP approximation is the main cause of the discrepancy in the magnitudes of the probabilities between GP/EP and SGD observed in the rest of the experiments in this chapter, and that for CE loss, the true Bayesian prior probabilities may in fact match the SGD probabilities a lot more closely. Furthermore, the MSE results suggest that MSE may give a good approximation to the Bayesian posterior probabilities (even the ones based on 0-1 likelihood). However, we note that this is a small toy model, and so our analysis leaves as an open question how to understand the error induced by the EP approximation in more realistic systems. One approach could be to cross-validate some of our results with state-of-the-art Monte Carlo sampling techniques for Bayesian inference [Rasmussen, 2004].

Description of figures. In Figures G.12, G.13 and G.14 in this section, we show, for three representative target Boolean functions, data comparing $P_{\text{SGD}}(f|S)$ for CE or MSE loss versus different ways to estimate $P_B(f|S)$, namely ABI, GP/EP for CE loss and GP/MSE for MSE loss.

In the **first column**, we show scatter plots comparing sampled probabilities (where functions not found in the sample are shown as if having a frequency of one). The colours denote the number of errors for each function on the test set.

The **second column** shows probability versus rank of the different test-set functions (when ranked by probability) for the two sampling methods.

The **third column** shows test accuracy histograms for the two sampling methods.

In the **first row**, we use sampling of parameters (and 0-1 loss) to estimate $P_B(f|S)$, which we use as the gold standard method as it has controlled small errors.

In the **second row**, we estimate $P_B(f|S)$ with the GP/EP approximation introduced in Section B.3.1.

In the **third row**, we compare $P_B(f|S)$ estimated from sampling of the exact MSE posterior (explained in Section B.3.2) versus the ABI sampling (for 0-1 loss).

In the **fourth row**, we compare $P_{SGD}(f|S)$ when training with MSE loss versus ABI sampling (for 0-1 loss).

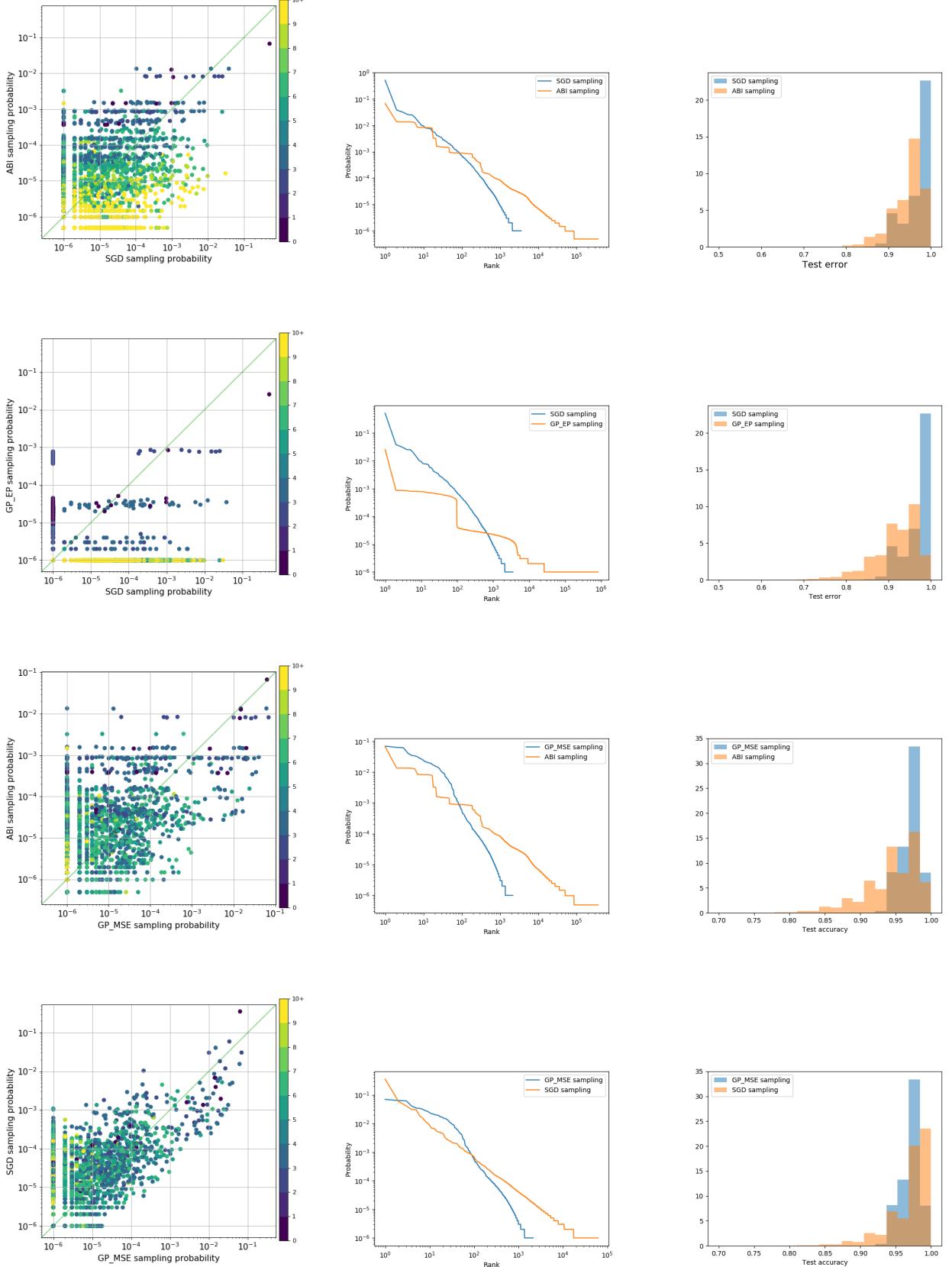


Figure G.12: Results for Section G.2 for target function with LZ complexity 35.0. Refer to text for detailed description. 270

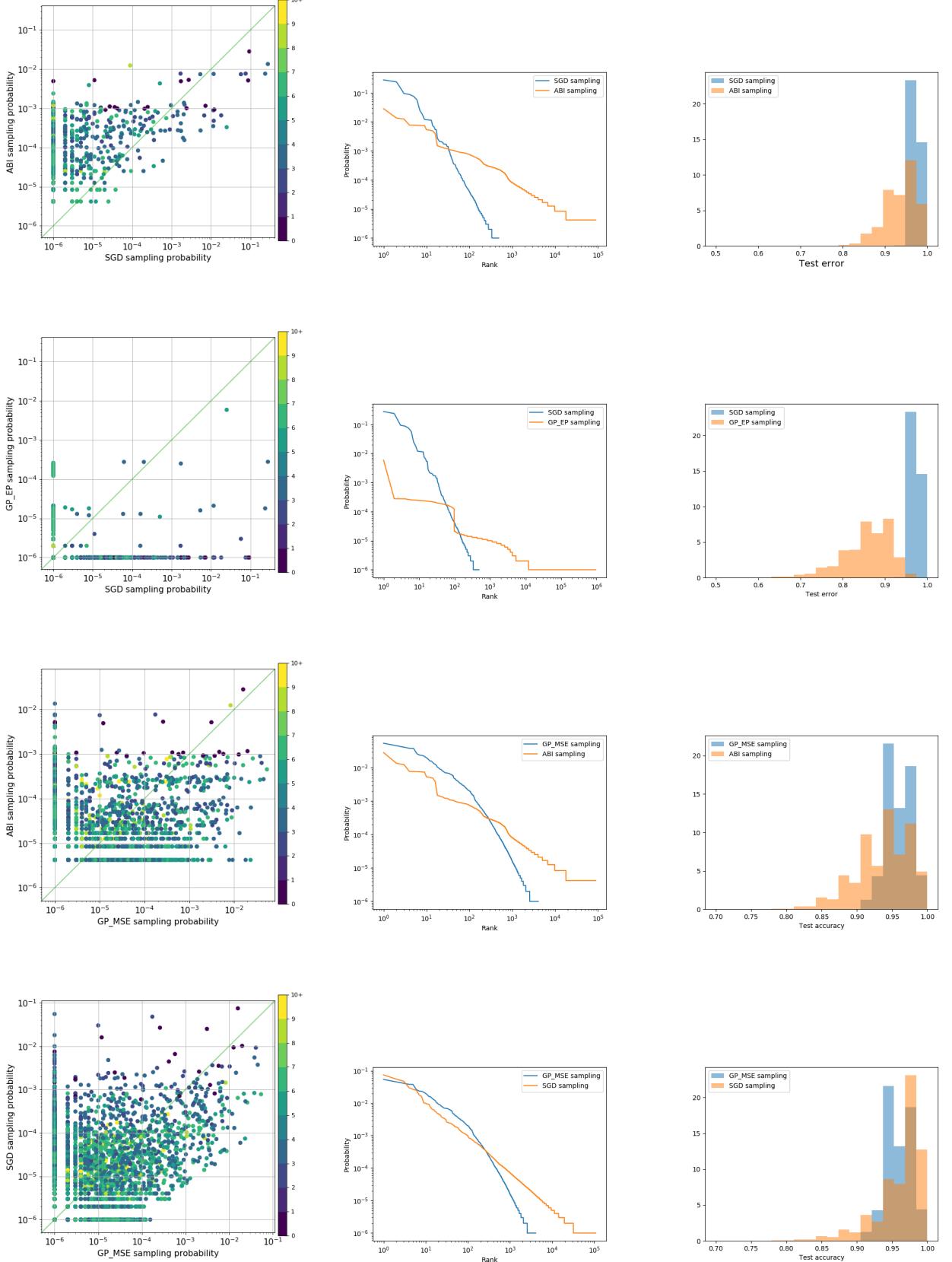


Figure G.13: Results for Section G.2 for target function with LZ complexity 28.0. Refer to text for detailed description. 271

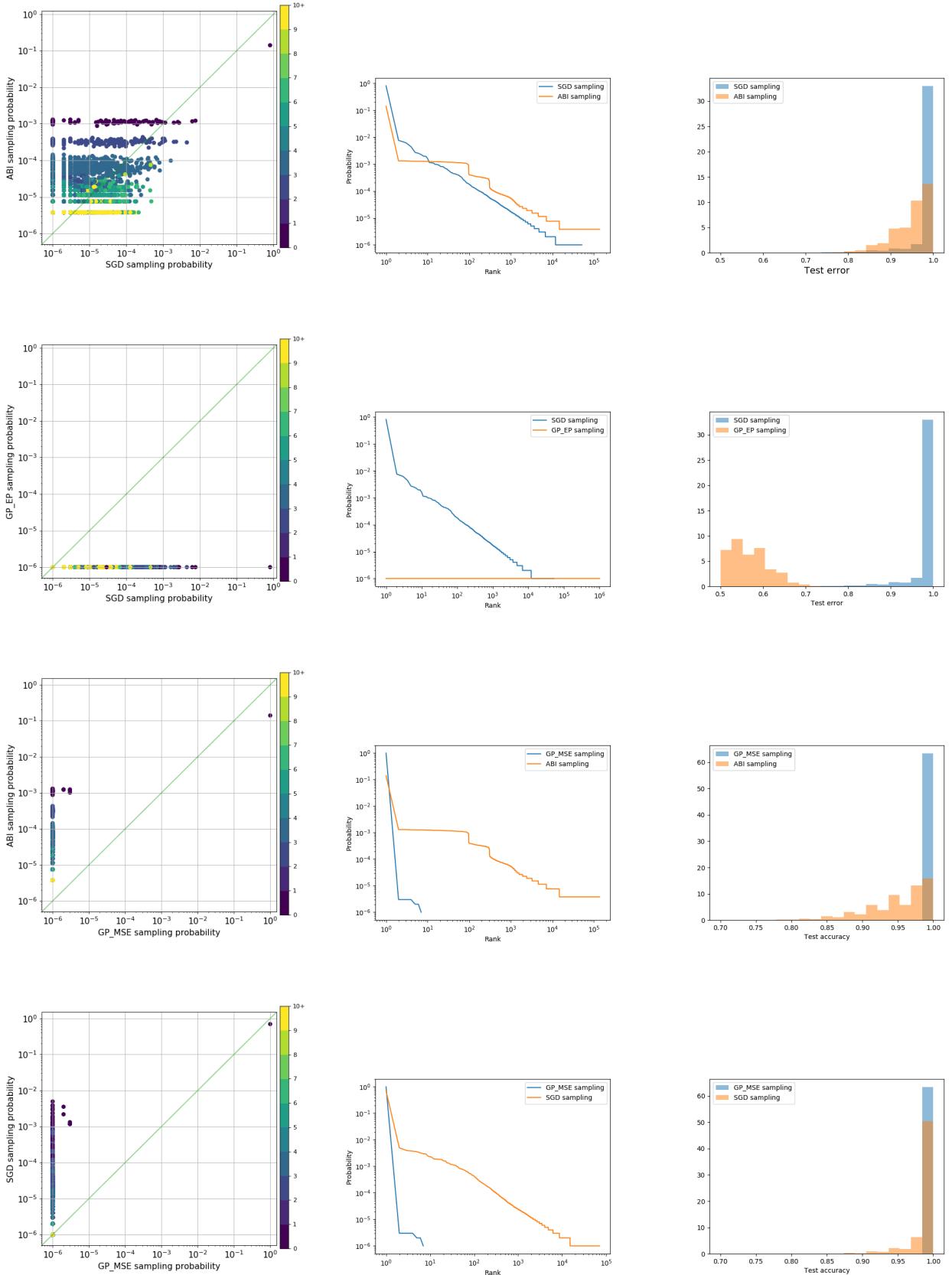


Figure G.14: Results for Section G.2 for a (different) target function of LZ complexity 28.0.
 Refer to text for detailed description.

G.3 Critical Sample Ratio

A measure of the complexity of a function, the critical sample ratio (CSR), was introduced in [Krueger et al., 2017b]. It is defined with respect to a sample of inputs as the fraction of those samples which are critical samples, defined to be an input such that there is another input within a box of side $2r$ centred around the input, producing a different output (for discrete outputs).

Following [Valle-Pérez et al., 2018], we use CSR as a rough estimate of the complexity of functions found in the posterior (conditioning on S), and the prior (i.e. functions on S). In our experiments, we used binarised MNIST with a training set size of 10000 and a test set of size 100 (analogously to the majority of our other experiments). For the prior, Figure G.15a, we randomly generated 100 functions with errors ranging between 0 and 10000 on S . For each function, we recorded the error, $P_B(f|S)$ and the CSR. For the posterior, we generated 500 functions with a range of errors on the test set and concatenated them with the function correct on the training set. We then proceeded as with the prior.

To calculate the CSR, we trained a DNN to model the function in question. Clearly this induces effects not purely due to the parameter-function map – although as we have seen in Section A.3, the functions found by SGD are likely to be similar to those that would be found by training by random sampling of parameters. Therefore, we may expect this process to approximately give the average CSR of parameters producing the function of interest. In Figure 2a. of [Valle-Pérez et al., 2018], an example of a very similar experiment with CIFAR10 can be found, where the network was not trained. This produces very similar results to our experiments with network training.

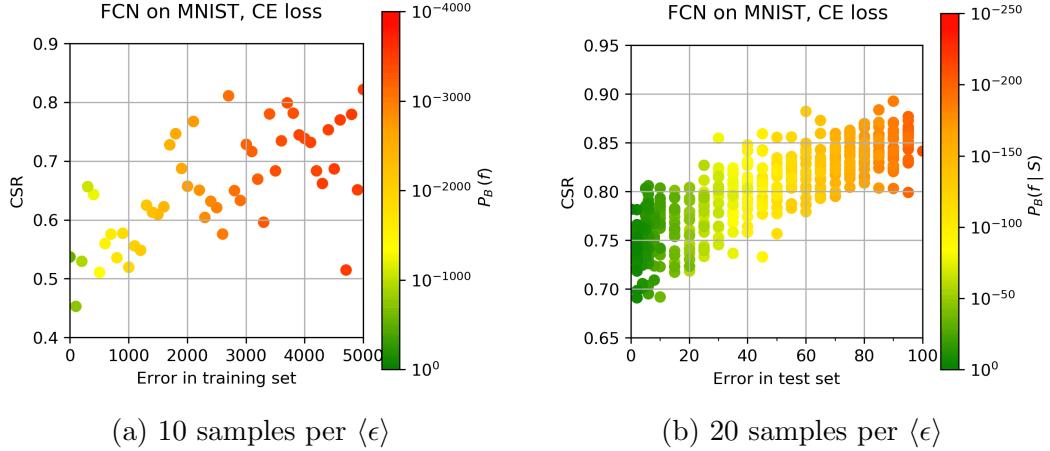


Figure G.15: (a) shows the result of our experiment on the functions in the prior; (b) shows the result out our experiment on functions in the posterior. Clearly there is a strong correlation between the CSR complexity and the error of the function; between the CSR complexity and $P_B(f|S)$; and between the error and $P_B(f|S)$. Different hyperparameters were used in the experiment to obtain a suitable range of complexity.

G.4 Notes on the distribution of MNIST data

Here, we show that p_i , the mean probability, for an FCN on MNIST with MSE loss function, that either SGD or the NNGP classifies the i 'th image in the test set E (used throughout this paper, with $|E| = 100$) incorrectly, varies by many orders of magnitude. We also show that images are classified in an approximately independent manner.

The values of p_i for the MNIST test set which is used in the majority of this paper are given in Figure G.16a. Note that $(1/n) \sum_i p_i = \langle \epsilon_G \rangle$. As can be seen, there are a small number of images that have a much higher probability of being miss-classified. It should be kept in mind that the exact spectrum of p_i will depend on which images are in the test set. For example, in Figure G.1 we compare two test sets (albeit with CE-loss and the EP approximation). As can be observed, the zero error function has significantly higher probability in the second test set, and the highest 1-error function has lower probability than for the test set we use in the paper. For test sets of this size such fluctuations are not unexpected.

Next we use the p_i to calculate probabilities for functions with other sets of errors. More specifically, consider test sets E_i containing only one image I_i (where i takes values in \mathbb{N}). Then calculating the probability of a function on $E = \{E_i\}_{i=0}^{i=|E|}$ for a test set of size $|E|$ can be done

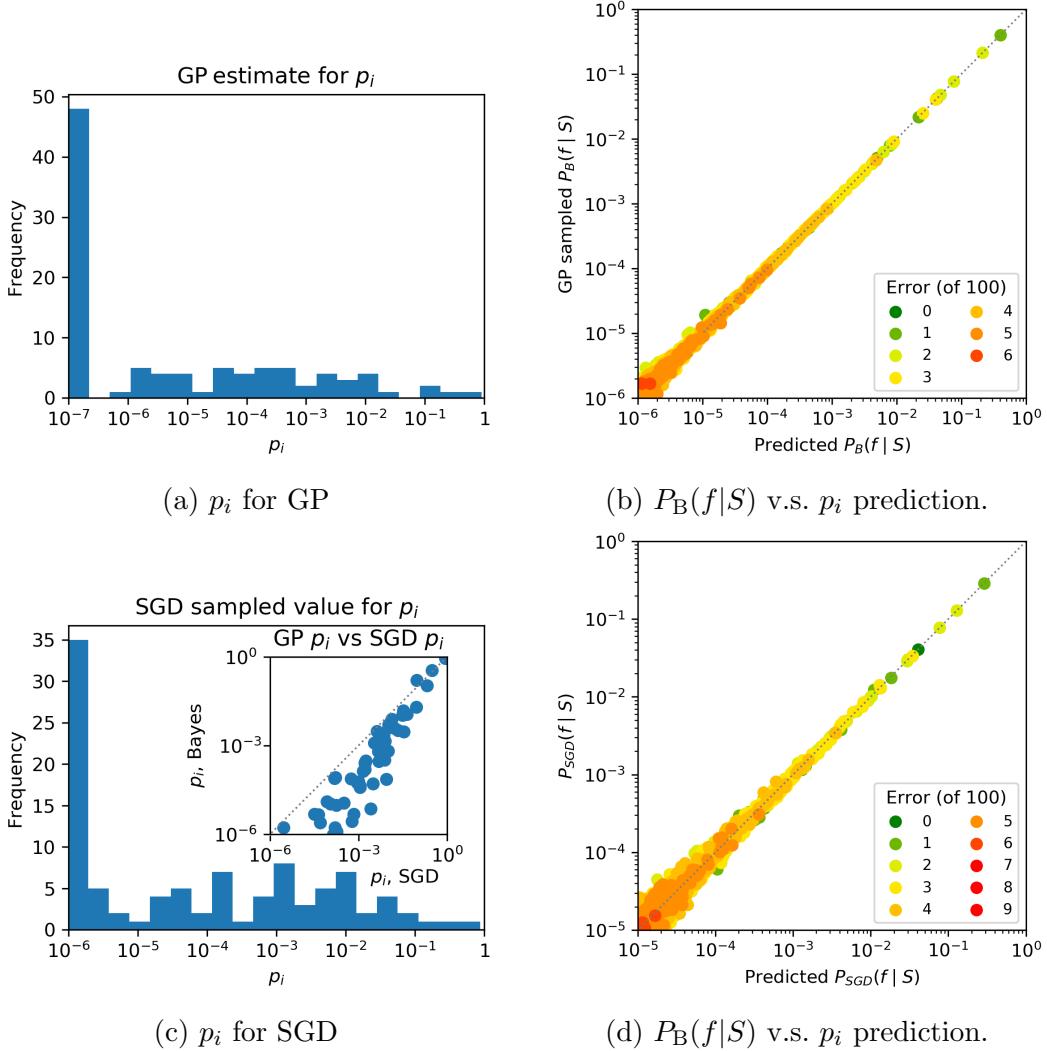


Figure G.16: (a) shows the NNGP estimate for the values of p_i for an FCN with MSE loss on MNIST, for the training set of size 10000 and test set of size 100 that we use in Chapter 6. Clearly these vary over many orders of magnitude. The sample size is 10^7 , so frequencies were cut off at 10^{-7} (so functions in that bin have $p_i \leq 10^{-7}$). 20 bins in total. (b) calculates the probability of other error functions using the values of p_i from Figure 6.1d using the assumption that the images from (a) are independently distributed, and compares this prediction for $P_B(f|S)$ to the value of $P_B(f|S)$ obtained by direct GP MSE sampling (see Figure 6.1d for the data). Each datapoint is for a specific function f , and clearly they are close to the $y = x$ line, implying that, at least for these higher probability functions found by direct sampling, the images in this small test set are classified by the GP in a (close to) independent fashion. Figures (c) and (d) are the equivalent of (a) and (b), but for SGD (see Figure 6.1a for the data). There were only 10^6 samples, so (c) and (d) are cut off at one order of magnitude lower than (a) and (b). (c) includes an inset comparing the values for p_i with GP MSE sampling from (a) and p_i with SGD from the main part of (c). The correlation is fairly tight for the highest p_i which dominate the total probability mass, and this correlation helps explain the strong correlation seen for other numbers of errors throughout this paper.

by multiplying the appropriate probabilities for functions on E_i . Applying the above method to this data is shown in Figure G.16b. All the functions shown are very close to the $y = x$ line, indicating that the images are classified in an approximately independent fashion.

In all our examples we observe a clear linear decay of the mean of $\log(P_B(f|S))$ v.s. ϵ_G . If the p_i were identical, then this would simply be what is expected for a Binomial distribution. For independent but different p_i over images in a finite test set the distribution is called the Poisson Binomial distribution. Obviously its values depend on the exact distribution of p_i . However, there exists a Chernoff bound

$$p(\epsilon) \leq P[\mathcal{E} > \epsilon] \leq \exp(-\epsilon(\log(\epsilon/\langle \epsilon_G \rangle) - 1) - \langle \epsilon_G \rangle), \quad (\text{G.1})$$

where $p(\epsilon)$ is the pmf, $P[\mathcal{E} > \epsilon]$ is the cmf, and $\langle \epsilon_G \rangle = (1/n) \sum_i p_i$ is the mean. On a log scale, this means

$$\log_{10}(p(\epsilon)) \leq [-\epsilon(\log(\epsilon/\langle \epsilon_G \rangle) - 1) - \langle \epsilon_G \rangle] \log_{10}(e), \quad (\text{G.2})$$

which indicates an exponential like drop-off for $\epsilon \geq \langle \epsilon_G \rangle$ (similar to what is observed in Figure 6.1a). Of course this is an upper bound and the actual distribution can strongly depend on the full spectrum of p_i values. We are currently exploring these issues in more detail.

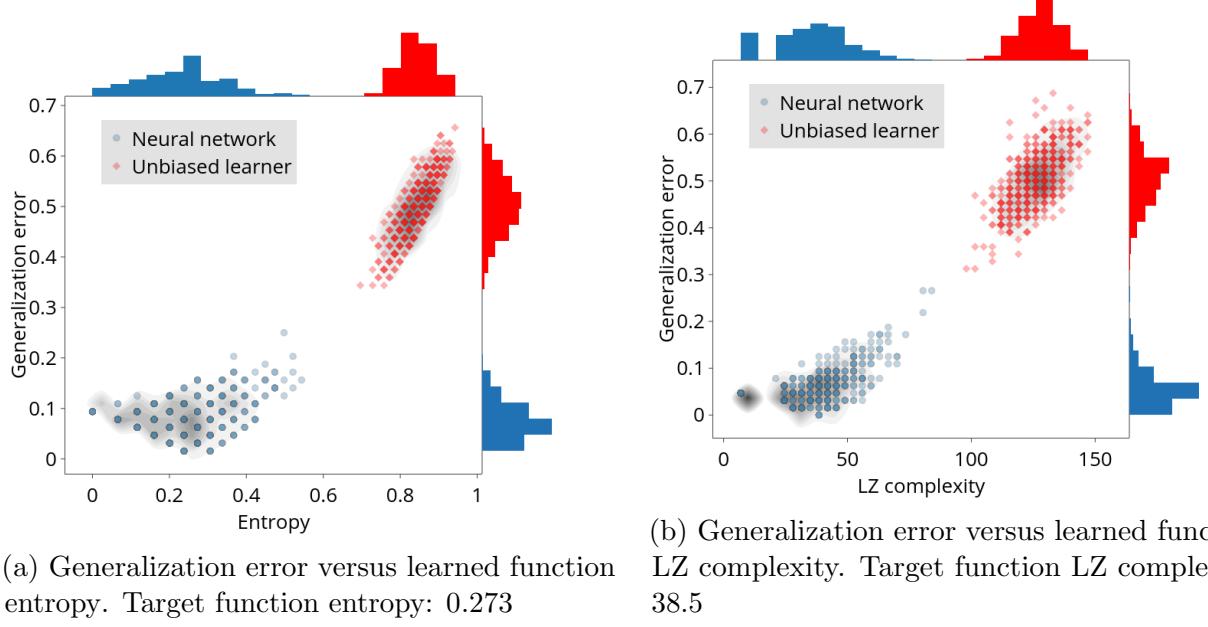
Appendix H

The error-complexity plane

In this chapter we show some results regarding the distribution of functions according to their generalization error and complexity, given a target function of a certain complexity. We study this question for the Boolean system introduced in Chapter 2 to help with the analysis.

H.1 The error-complexity landscape

Figure H.1 suggests that even for a fixed target function, if it is simple, complexity correlates with generalization error. Here complexity refers to that of a candidate function, and the generalization error is for this candidate relative to the target. This correlation gives a very direct way of seeing that bias towards simple functions helps to learn functions with low generalization error, for simple target functions. To provide some theoretical support, we derive bounds for the region in the error-complexity plane that contains all possible Boolean functions. When using entropy as the complexity measure, we can get a very precise picture. We can get a more qualitative but still insightful picture using Kolmogorov complexity.



(a) Generalization error versus learned function entropy. Target function entropy: 0.273

(b) Generalization error versus learned function LZ complexity. Target function LZ complexity: 38.5

Figure H.1: Generalization error versus different complexity measures for learned functions, for 500 random initialization and training sets of size 64, for a fixed target function. The histograms show the frequency of generalization errors and complexities. The blue circles and blue histograms correspond to the (7, 40, 40, 1) neural network, and the red dots and histograms to an unbiased learner which also fits the training data. The black histogram shows the density of dots (darker is higher density). The network is trained using the advSGD algorithm (see Section A.1). Generalization error is defined with respect to off-training set samples.

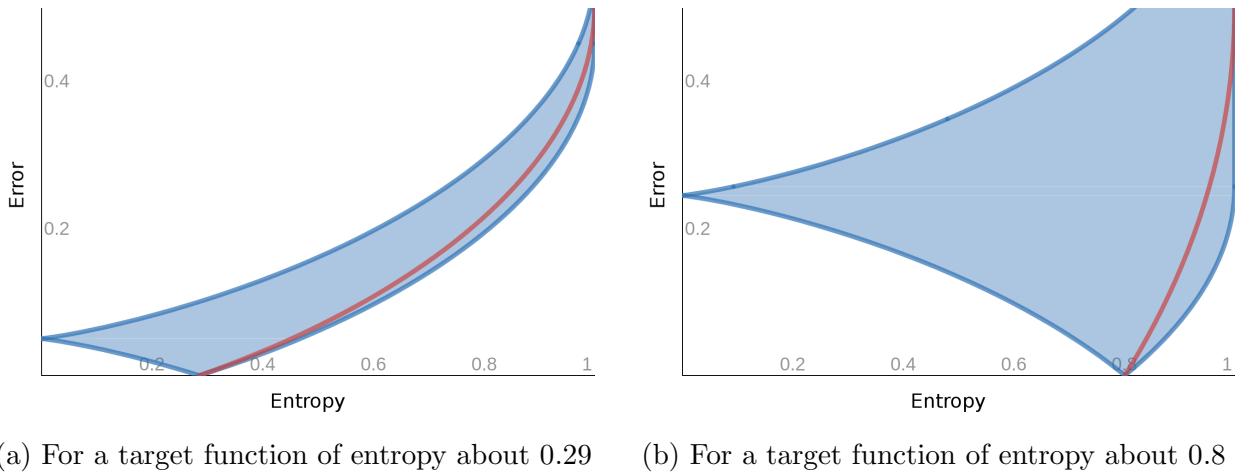
As we are not considering a training set here, we cannot define an off-training generalization error (which we use in the experiments in Chapter 2). Therefore, we define *error* here as the fraction of bits in which a function differs with the fixed target function which, if uniformly sampling the input space, corresponds to the standard notion of i.i.d. generalization error [Wolpert and Waters, 1994].

H.1.1 Error-entropy plane

In Section H.3, we derive a series of inequalities relating the generalization error and the entropy of a function, which delineate the region where all possible functions lie, for a target function of fixed entropy. As can be seen in Figure H.2, the shape of this region forces the entropy and error to correlate. The distribution of functions in this region is very biased towards high entropy. In the figure, we show (red line) the average entropy of the functions that produce a

given error value. Within this line, the distribution is itself exponentially concentrated near the 0.5 error point. This is the fundamental reason why the unbiased learner gives functions with high error (Figure H.1).

For the neural network, on the other hand, we see points concentrated at lower entropy and error, and we can even see the ‘V’ shape around the entropy of the target function predicted by the bounds.



(a) For a target function of entropy about 0.29 (b) For a target function of entropy about 0.8

Figure H.2: The blue lines indicate the error-entropy bounds for all possible Boolean functions, given a fixed target function, as derived in Section H.3. The region is symmetric with respect to reflection along the error equals 0.5 line. The red line shows where the number of functions exponentially concentrates, for a fixed error, as the size of the input space grows to infinity.

H.1.2 Error-Kolmogorov complexity plane

In Section H.4, we also derive bounds relating the Kolmogorov complexity of a function and its error, relative to a fixed target function. If we ignore the order one terms (which we can in the asymptotic limit of the input space size going to infinity), we get regions like those shown in Figure H.3. The upper boundary of the region is only valid for almost all functions (see Section H.4). We can see that the shape of the region also imposes a correlation between complexity and error, for simple target functions.

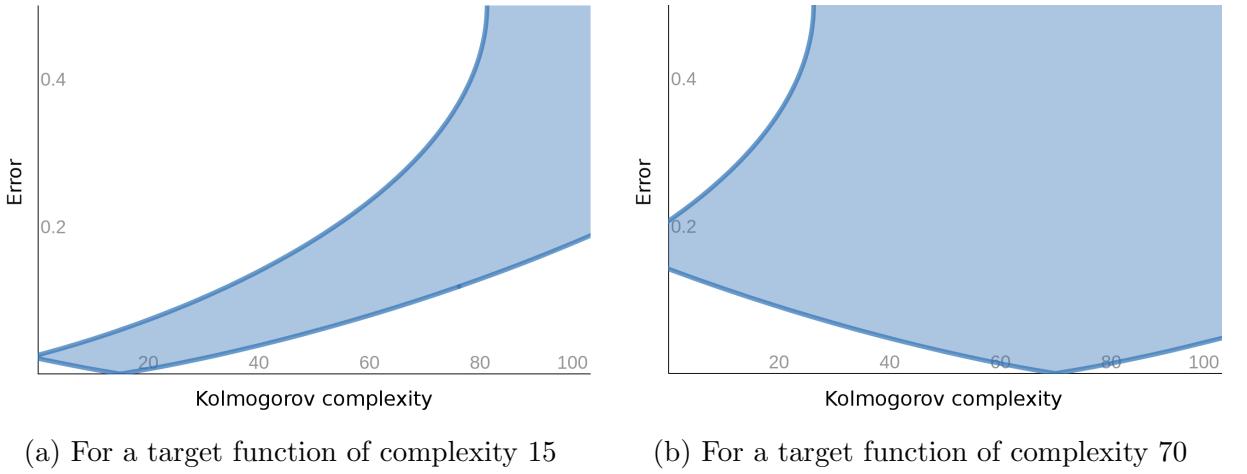


Figure H.3: The blue region contains the error-Kolmogorov complexity pairs for almost all possible functions (for each value of error), given a fixed target function, corresponding to the bounds derived in Section H.4.

H.2 What functions does the network find?

Knowing the error-complexity landscape for all possible functions is only half the story. We also want to know which functions are more likely to be produced by a learning algorithm. The results shown in Figure H.1 give us a good picture of this. In this section, we show a simple model which assumes a bias depending only on entropy, and which captures some of the behaviour observed for neural networks. We also show empirical results showing the complexities of the functions actually found by the network.

In Figure H.4, we show the probability that a learning algorithm chooses a function of a certain entropy and error, given a target function of entropy 0.27, for two algorithms. Figure H.4b shows the unbiased learner introduced before, and Figure H.4a shows a learner where the probability of picking a particular function (assuming it is consistent with the training set) decreases exponentially with the function entropy. In Section H.5, we derive exact expressions for the probability of choosing a function of a given entropy and generalization error, for both the biased and unbiased learners, which we used to get the results in Figure H.4.

We can see that, as expected for this simple target function, the unbiased learner will typically find high-entropy functions with high generalization error, while the biased learner is more likely to find simpler functions with lower generalization error, agreeing with the results

in Figure H.1. Furthermore, the biased algorithm shows a linear correlation between the target function entropy and learned function entropy, which also agrees with the results for the neural network (Fig. H.5a).

Although, the algorithm with entropy-dependent bias shows similar behaviour to the neural network when only considering the entropy of the functions, it does not capture the full bias, as evidenced by the clear outliers in Figure D.2c, and results like those in Figure 2.4 in Chapter 2. Figure H.5b shows that for some functions the neural network is able to generalize significantly better than the purely entropy-dependent bias predicts. Despite these differences, the entropy bias offers an interesting toy model to understand the effects of simplicity bias on learning, and could offer a starting point for more advanced analysis.

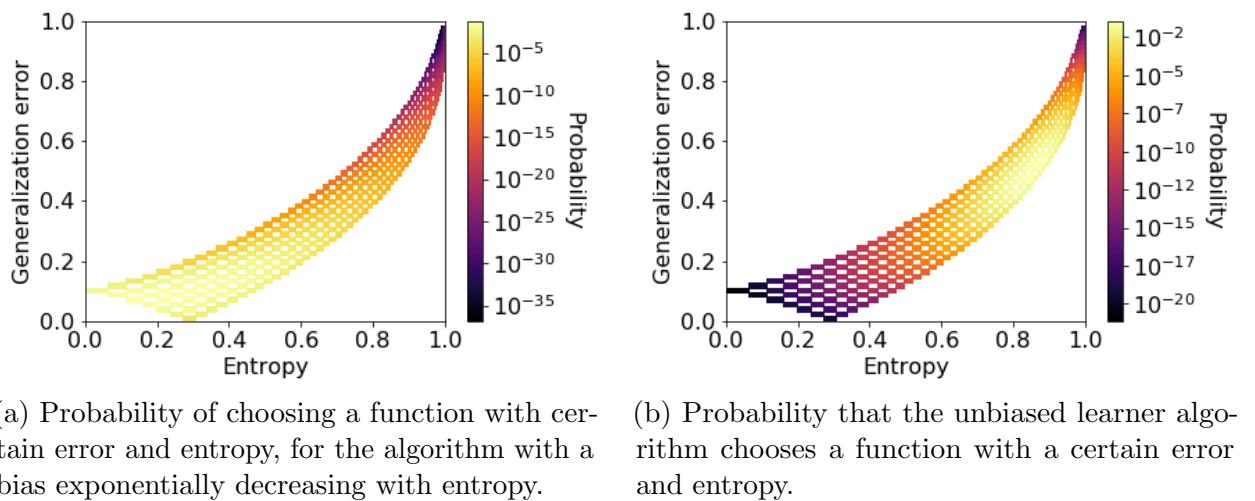


Figure H.4: Probability of choosing a function with certain error and entropy, for the biased and unbiased algorithm. Note that for the biased learner (plot (a)) high probability is found at the low error and low entropy corner, while for the unbiased learner (plot (b)), high probability is found at the high error and high entropy corner. The target function has entropy 0.27. Note that the checkerboard pattern simply comes from the combinatorics of binary strings disallowing certain pairs of error-entropy (see Section H.3).

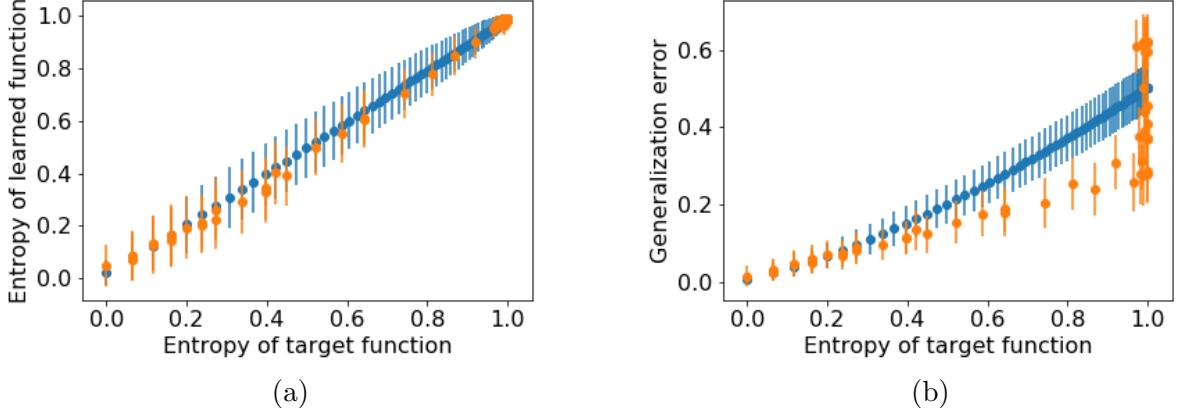
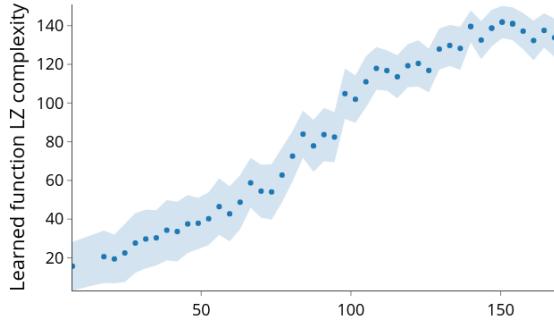
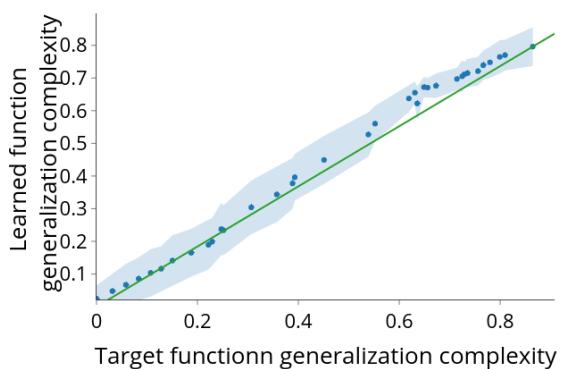


Figure H.5: Learned function entropy and generalization error versus entropy of target function. Dots are means, and lines correspond to standard deviations. Orange dots and lines are for a neural network of shape $(7, 40, 40, 1)$ trained with advSGD, averaged over 500 random initializations and training sets. The blue dots and error bars are calculated for the learner with exponential entropy bias.

In Figure H.6, we can see how the LZ and generalization complexity of the learned function depends on the complexity of the target, showing a similar direct relationship to the one seen for entropy. Overall, we see that although the bias of the network causes the network to prefer simpler functions, the functions that the network learns grow in complexity with the complexity of the target function. This is probably due to a combination of the bias not being too strong, and the fact that typically functions which are too simple functions are typically incompatible with a training set produced by a complex target function. Regarding this, in Section H.6, we prove a lower bound on the complexity of the simplest function compatible with the training set, holding with high probability. However, the bound is probably not very tight, if we compare it with the results in Figure H.6, so a better theoretical understanding of this behaviour is still needed.



(a)



(b)

Figure H.6: Complexity of learned functions versus the complexity of the target function, for a network of shape $(7, 40, 40, 1)$. The dots are means, and the shaded area a linear interpolation of the standard deviation, over 500 random initializations and training sets, training using advSGD.

The results in Figure H.6 give another way to understand the complexity-dependence of generalization, which is similar to the argument for simple Occam algorithms Blumer et al. [1987], Wolpert and Waters [1994]. For complex target functions, the network is effectively considering a larger class of functions. This is because the set of functions fitting the data is composed of complex functions and, among these, the simplicity bias does not have a large effect, as there is a smaller range of probabilities. This intuition is formalized by the PAC-Bayes theorem.

H.3 Bounds and number of functions in the error-entropy plane

Here we represent Boolean functions (through their table of outputs) as a string of bits, of length n . The entropy of a bit string with n_1 1s and $n_0 = n - n_1$ 0s is defined in Section D.1.1. Consider a target function with \tilde{n}_1 1s and \tilde{n}_0 0s. Consider another function which differs in r bits with the target function, and which has n_1 1s and n_0 0s. Furthermore, call the number of bits in which it differs with the target, and for which the target bit was 1, r_1 , and similarly for r_0 . We have $r_0 + r_1 = r$. It is clear that the number of 1s and 0s satisfy

$$n_1 = \tilde{n}_1 + (r_0 - r_1) = \tilde{n}_1 + r - 2r_1 \quad (\text{H.1})$$

$$n_0 = \tilde{n}_0 + (r_1 - r_0) = \tilde{n}_0 + r - 2r_0 \quad (\text{H.2})$$

We also satisfy these inequalities

$$0 \leq r_0 \leq r$$

$$r_0 \leq \tilde{n}_0$$

$$r - r_0 = r_1 \leq \tilde{n}_1$$

which translate to these inequalities for n_0 :

$$\tilde{n}_0 - r \leq n_0 \leq \tilde{n}_0 + r$$

$$n_0 \geq r - \tilde{n}_0$$

$$n_0 \leq 2n - r - \tilde{n}_0$$

These inequalities on n_0 can be easily translated to inequalities on the entropy, noting that entropy is a monotonically increasing function of n_0 for $0 \leq n_0 \leq \frac{n}{2}$ and monotonically

decreasing for $\frac{n}{2} \leq n_0 \leq 1$. These inequalities, where the error is normalized and define to be $\epsilon = \frac{r}{n}$, define the region shown in Figure H.2.

The other quantity of interest, used in Section H.2, is $N_{S_t}(\epsilon, S)$, the number of functions with a given error ϵ and entropy S , for a target with entropy S_t . We can just work out the number $N_{\tilde{n}_0}(r, n_0)$, for a given r and n_0 , for a target with \tilde{n}_0 0s, knowing that there are two values of n_0 producing a given value of S . Using Eq. H.2,

$$N_{\tilde{n}_0}(r, n_0) = \binom{\tilde{n}_0}{r_0} \binom{\tilde{n}_1}{r_1}, \quad (\text{H.3})$$

where $r_0 = \frac{\tilde{n}_0 - n_0 + r}{2}$, $\tilde{n}_1 = n - \tilde{n}_0$, and $r_1 = r - r_0$. We can use the asymptotic form of the binomial coefficients [1] to get

$$\log_2 N_{\tilde{n}_0}(r, n_0) \sim \tilde{n}_0 H(r_0/\tilde{n}_0) + \tilde{n}_1 H(r_1/\tilde{n}_1), \quad (\text{H.4})$$

where $H(x) = -x \log_2 x - (1-x) \log_2 (1-x)$ is the binomial entropy function. If we write $r_0 = ar$, $r_1 = (1-a)r$, and allow a to vary continuously between 0 and 1, we can look for a maximum of the number of functions, for fixed r , by setting the derivative of the expression in Eq. H.4 with respect to a , and setting it to 0. Doing that gives $r_0/\tilde{n}_0 = r_1/\tilde{n}_1 = \epsilon$. Now $r = r_1 + r_0 = \epsilon(\tilde{n}_0 + \tilde{n}_1) = \epsilon n$, so that $\epsilon = r/n$ is just the (normalized) error. This points gives a value of r_0 , and therefore n_0 and S , where $N_{\tilde{n}_0}(r, n_0)$ exponentially concentrates, for a fixed ϵ . This is what is plotted as the red line in Figure H.2. Finally, for points in this line, we have

$$N_{\tilde{n}_0}(r, n_0) = 2^{nH(\epsilon)+o(n)}, \quad (\text{H.5})$$

so we can see that the number of functions also exponentially concentrates at the point $\epsilon = 0.5$, where $H(\epsilon)$ is maximum (and equal to 1).

H.4 Bounds for the error-Kolmogorov complexity plane

Consider a target function f_t , and another function f , both over n inputs (so represented as binary strings of length n), which differ from each other in r bits. Then we can describe f by giving f_t and specifying the locations of the r bits and their values, giving the following bound

on the Kolmogorov complexity of f :

$$K(f) \leq K(f_t) + \log_2 \binom{n}{r} + r + O(1) \quad (\text{H.6})$$

We have a similar bound with f and f_t swapped, so that we have

$$|K(f) - K(f_t)| \leq \log_2 \binom{n}{r} + r + O(1) \quad (\text{H.7})$$

On the other hand, giving a f_t and a f which differ in r bits describes a binary string of length n with r bits, furthermore there is a one-to-one correspondence between f and this binary string, for a given f_t . So if the string is s , we know $K(s) \leq K(f) + K(f_t) + O(1)$. However, from a simple counting argument Ming and Vitányi [2014], we know that most such binary strings have complexity close to $\binom{n}{r}$, so that for most f , given fixed f_t and r , we have

$$K(f_t) + K(f) \geq \binom{n}{r} + O(1) \quad (\text{H.8})$$

Ignoring the $O(1)$ terms is what gives the bounds in Figure H.3.

H.5 Probability of choosing a function of given error and entropy

To calculate the probability of choosing any function of a particular error and entropy (Section H.2), we first calculate the probability of choosing a particular function which differs with the target in r bits (which we call the *error*), and has entropy S .

We assume sampling the input space (of cardinality n) uniformly at random without replacement, and a sample of size m . Therefore, every particular sequence of m instances¹ has the same probability $\frac{(n-m)!}{n!}$. The probability that our sample contains a particular set of m instances in any order is then $\frac{m!(n-m)!}{n!} = 1/\binom{n}{m}$.

Consider a particular function with error r . For it to be consistent with the sample, the sample must contain only instances within the set of $n - r$ instances on which the function

¹refer to elements of the input space as *instances*

agrees with the target. There are $\binom{n-r}{m}$ such sets of size m , so the probability that the sample is consistent with this particular function is $(\binom{n-r}{m}) / (\binom{n}{m})$.

With any such sample, there are 2^{n-m} functions consistent with it (as we are assuming the hypothesis class to be all possible Boolean functions over the input space). The unbiased learner chooses any of these with equal probability $\frac{1}{2^{n-m}}$, and so the probability that it chooses the particular function is $(\binom{n-r}{m}) / (\binom{n}{m}) \frac{1}{2^{n-m}}$

Because the events corresponding to choosing particular functions are mutually exclusive, the probability $P(r, S)$ of choosing any function of error r and entropy S for the unbiased learner is simply the sum of the probabilities of all such functions:

$$P_{S_t}(r, S) = N_{S_t}(r, S) \frac{\binom{n-r}{m}}{\binom{n}{m}} \frac{1}{2^{n-m}}, \quad (\text{H.9})$$

where $N_{S_t}(r, S)$ is the number of functions with error r and entropy S , for target function entropy S_t , given by Eq. H.3. Note that we take the convention that $\binom{n}{k} := 0$ when $k > n$ or $k < 0$.

For a learner with bias $P(S) = 2^{-S}$ (we absorb the factor $|X|$ in the definition of S for simplicity), the probability $P(f)$ that it chooses a particular function f given a particular training set of size m is no longer $\frac{1}{2^{n-m}}$. Instead it is given by

$$P(f) = \frac{2^{-S(f)}}{\sum_{g \in U} 2^{-S(g)}},$$

where $S(f)$ is the entropy of function f , and U is the set of all functions compatible with the training set. We want to compute the quantity in the denominator. Call n'_1 the number of instances in the training set where the target (and f) equals 1. Then there are $\binom{n-m}{n_1 - n'_1}$ functions in U with n_1 1s, and so the numerator is:

$$N(n'_1) := \sum_{g \in U} 2^{-S(g)} = \sum_{n_1=n'_1}^n \binom{n-m}{n_1 - n'_1} 2^{-S(n_1)}, \quad (\text{H.10})$$

where $S(n_1)$ is the entropy of a bit string with n_1 1s.

What is the probability of a training set with a particular value of n'_1 ? We can use the result from Section H.3 that the number of differing bits which are equal to 1, r_0 , is fixed by the

entropy of the target and the entropy of f , from Eq. H.2. The number of training sets of size m consistent with f with a particular value of n'_1 is then $\binom{n_1 - r_0}{n'_1} \binom{n_0 - r_1}{m - n'_1}$, and the probability of each of them is $1/\binom{n}{m}$ as before. Putting everything together the probability of obtaining any function of error r and entropy S , for a target function with entropy S_t , and training set size m , is

$$P_{S_t}(r, S) = N_{S_t}(r, S) \sum_{n'_1=0}^m \frac{\binom{n_1 - r_0}{n'_1} \binom{n_0 - r_1}{m - n'_1}}{\binom{n}{m}} \frac{2^{-S}}{N(n'_1)}, \quad (\text{H.11})$$

where $N_{S_t}(r, S)$ is given by Eq. H.3, $N(n'_1)$ is given by Eq. H.10, $r_0 = \frac{\tilde{n}_0 - n_0 + r}{2}$ (from Eq. H.2, where \tilde{n}_0 and n_0 are the number of 0s corresponding to entropy S_t and S , respectively), $r_1 = r - r_0$, and n_1 is the number of 1s corresponding to entropy S .

Finally, note that because the number of instances in the training set is fixed to m (as we sample without replacement), the off-training set generalization error for functions consistent with the training set is just $r/(n - m)$.

H.6 Lower bound on the complexity of functions compatible with the training set

We first prove the following theorem which shows that you cannot have functions with a big error, and which are very simple, with high probability over training sets (when the sampling distribution over the input space is uniform)

Theorem H.6.1. *Let n be the size of the input space. If for a fraction greater than $1/2^\alpha$ of possible training sets of size m drawn from a fixed target function, we have that there exists some function f consistent with the training set, with error r with respect to the target, and with complexity $K(f) \leq C$, then*

$$C \geq \log_2 \binom{n}{m} - \log_2 \binom{n - r}{m} - \alpha$$

Proof. The set S of all functions with complexity $K(f) \leq C$ has cardinality at most 2^C . There are $\binom{n}{m}$ training sets with m unique elements. We assume that for $2^{-\alpha} \binom{n}{m}$ of these, there is

a function $f \in S$ which is consistent with it. We call N_f the number of training sets with which a particular $f \in S$ is consistent. The mean of N_f is at least $2^{-\alpha-C} \binom{n}{m}$, and so the maximum N_f^* is also at least this number (as the maximum is always greater than the mean). By assumption the f^* corresponding to N_f^* differs with the target in r bits. Therefore, the training sets consistent with it must avoid these bits. There are $\binom{n-r}{m}$ such training sets, and so $N_f^* \leq \binom{n-r}{m}$. Combining this with the previous inequality gives $2^{-\alpha-C} \binom{n}{m} \leq \binom{n-r}{m}$, and taking the logarithm of both sides, gives the desired result. \square

If $r = \epsilon n \ll n$, and $m \ll n$, then the bound is approximately $C \geq m\epsilon - \alpha$, which is what one gets from a simple PAC bound with hypothesis class of size 2^C . Theorem H.6.1 is essentially a rephrasing of the PAC bound for training sets with m distinct elements (that is sampling without replacement).

If we combine the bound from Theorem H.6.1 with those in Figure H.3, we get a lower bound on the complexity of the simplest function compatible with the training set, holding with high probability. This bound depends on the complexity of the target, because of the dependence in Eq. H.7. The result is plotted in Fig. H.7.

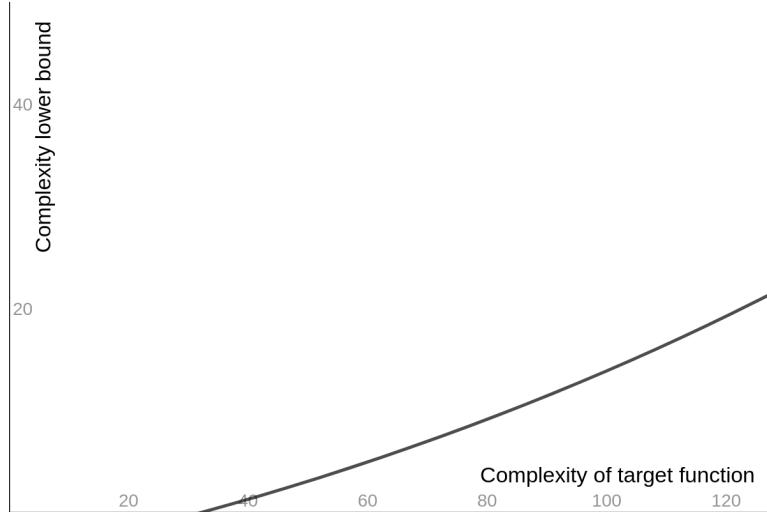


Figure H.7: Lower bound on the complexity of the simplest function compatible with the training set, versus the complexity of the target function. This is found by intersecting the bound from Theorem H.6.1, with the bound from Eq. H.7.

H.7 Error-complexity histograms

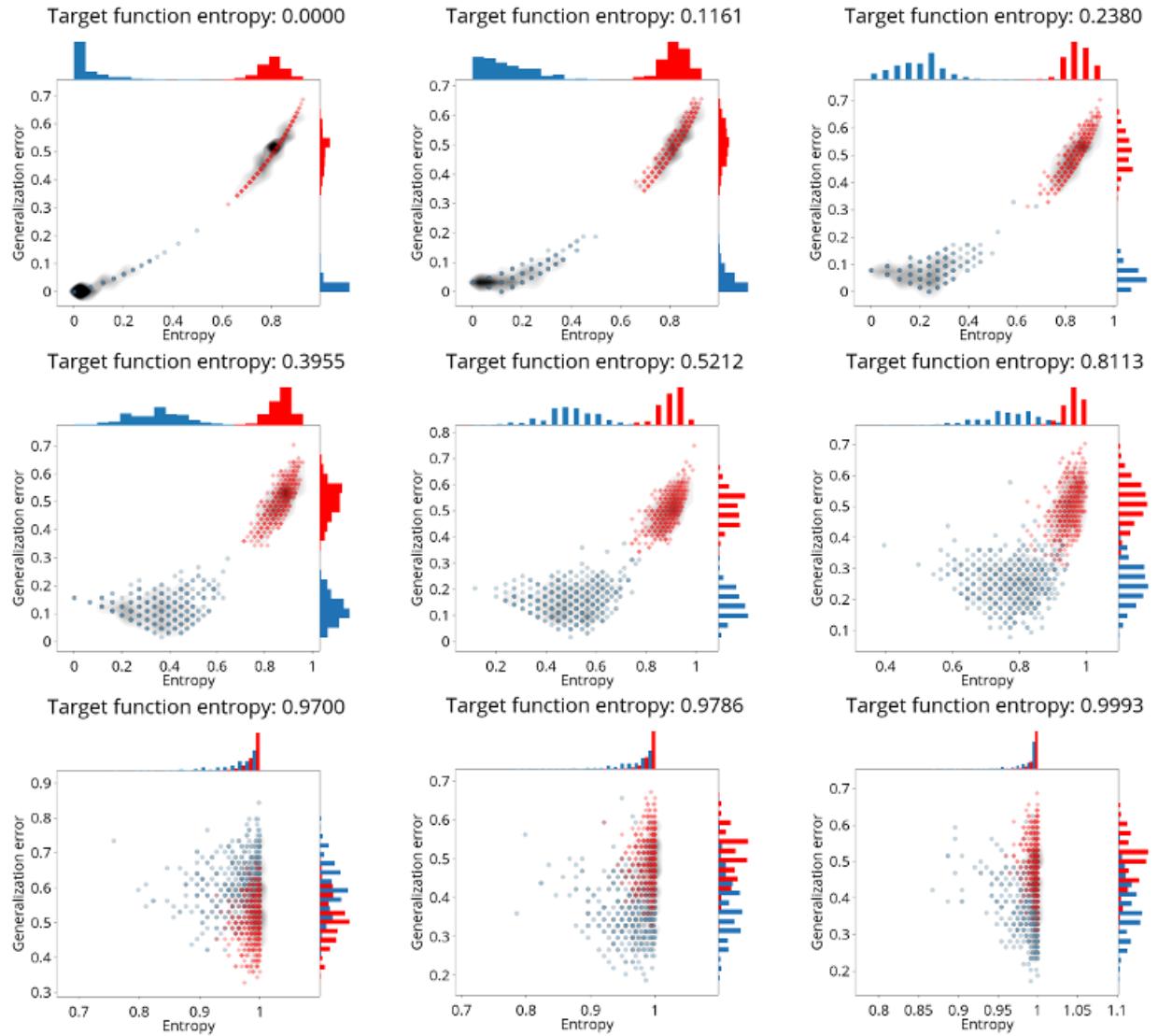


Figure H.8: Generalization error versus entropy. Red dots and histograms correspond to the unbiased learner, blue dots and histograms to the neural network of shape $(7, 40, 40, 1)$.

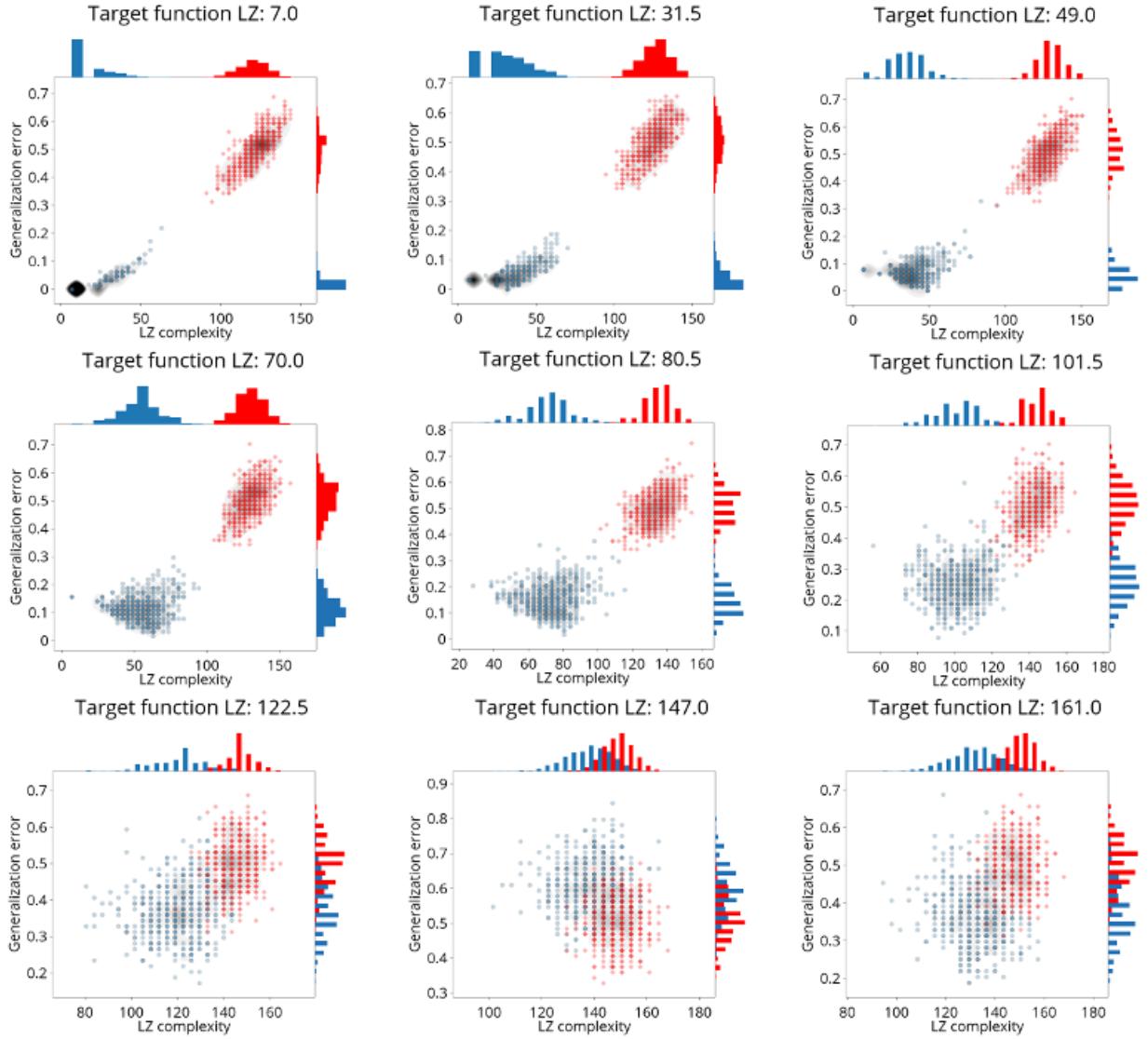


Figure H.9: Generalization error versus LZ complexity. Red dots and histograms correspond to the unbiased learner, blue dots and histograms to the neural network of shape $(7, 40, 40, 1)$.

As can be seen in Figure H.1, for simple target functions the neural network finds solutions which are significantly simpler, and generalize better than the functions found by the unbiased algorithm. Furthermore, as can be seen in Figure 2.3a (and in Section H.7), although the error increases with the complexity of the target function, the network still generalizes significantly better than the unbiased learner, as long as the target function does not have maximum complexity (corresponding to random functions).

As we can see in Figure H.8 in Section H.7, for some target functions with maximum entropy

(but which are simple when measured using LZ complexity), the network still generalizes better than the unbiased learner, showing that the bias towards simpler functions is better captured by more powerful complexity measures than entropy (LZ is a better approximation to Kolmogorov complexity than entropy Cover and Thomas [2012]). This is confirmed by experiments where we fix the target function entropy (to 1.0), and see that the generalization error still shows considerable variation, and correlation with complexity (Fig 2.4 in Section 2.4.1).

Bibliography

- Leo Breiman. Reflections after refereeing papers for nips. *The Mathematics of Generalization*, pages 11–15, 1995.
- Lenka Zdeborová. Understanding deep learning is also a job for physicists. *Nature Physics*, pages 1–3, 2020.
- Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.
- Alan Mathison Turing. Intelligent machinery, 1948.
- Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- VV Ashby. R.(1952) design for a brain, 1952.
- Friedrich August Hayek. *The sensory order: An inquiry into the foundations of theoretical psychology*. University of Chicago Press, 1952.
- AM Uttley. Conditional probability machines and conditioned reflexes. inautomata studies. eds. ce shannon and j. mccarthy, 1956.
- Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

- RD Joseph. On predicting perceptron performance. In *PROCEEDINGS OF THE INSTITUTE OF RADIO ENGINEERS*, volume 48, pages 398–398, 1960.
- Dale Raymond Lumb. An evaluation of the perceptron theory. 1959.
- James S Bryan. Experiments in adaptive pattern recognition. *IEEE Transactions on Military Electronics*, (2 & 3):174–179, 1963.
- A Gamba. Optimum performance of learning machines. *PROCEEDINGS OF THE INSTITUTE OF RADIO ENGINEERS*, 49(1):349, 1961.
- G Palmieri and R Sanna. *Automatic Probabilistic Programmer: Analyzer for Pattern Recognition*. Feltrinelli, 1960.
- Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.
- Bernard Widrow. Generalization and information storage in network of adaline'neurons'. *Self-organizing systems-1962*, pages 435–462, 1962.
- Claude E Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723, 1938.
- John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.
- Richard Lewis Mattson. *The design and analysis of an adaptive system for statistical classification*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, 1959.
- RA Mattson. A self-organizing logical system. 1959 eastern joint computer conf. *Conv. Rec. Inst. Radio Engrs, NY*.
- AG Ivakhnenko and VG Lapa. Cybernetic predictive devices, 1965.
- Minsky Marvin and A Papert Seymour. Perceptrons, 1969.

Mikel Olazaran. A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26(3):611–659, 1996.

Harvey A Cohen. The perceptron controversy. URL <http://harveycohen.net/image/perceptron.html>.

Jim Howe. Artificial intelligence at edinburgh university: A perspective. *Archived from the original on*, 17, 2007.

Russell Stuart, Norvig Peter, et al. Artificial intelligence: a modern approach, 2003.

William A Little. The existence of persistent states in the brain. *Mathematical biosciences*, 19(1-2):101–120, 1974.

S-I Amari. Neural theory of association and concept-formation. *Biological cybernetics*, 26(3):175–185, 1977.

Teuvo Kohonen. Correlation matrix memories. *IEEE transactions on computers*, 100(4):353–359, 1972.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Haim Sompolinsky. Statistical mechanics of neural networks. *Physics Today*, 41(21):70–80, 1988.

Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Doklady Akademii Nauk*, volume 114, pages 953–956. Russian Academy of Sciences, 1957.

Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7, 1970.

PJ Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences, Harvard University*. PhD thesis, Masters Thesis, 1974.

Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

DB Parker. Learning-logic (tr-47). *Center for Computational Research in Economics and Management Science. MIT-Press, Cambridge, Mass*, 8, 1985.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Kunihiro Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron. *IEICE Technical Report, A*, 62(10):658–665, 1979.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

Naftali Tishby. Statistical physics models of supervised learning. In *The Mathematics Of Generalization*, pages 215–242. Addison-Wesley Longman Publishing Co., Inc., 1995.

Frank J Smieja. Learning and generalization in feed-forward neural networks (neural networks, backpropagation learning algorithm). 1989.

DJ Wallace. Neural network models: a physicist’s primer. *Computational Physics (SUSSP 32)*, pages 168–211, 1987.

Vladimir Vapnik. On the uniform convergence of relative frequencies of events to their probabilities. In *Doklady Akademii Nauk USSR*, volume 181, pages 781–787, 1968.

Vladimir Vapnik and Alexey Chervonenkis. Theory of pattern recognition, 1974.

Vladimir N Vapnik. The nature of statistical learning theory, 1995.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

David H Wolpert. Mathematics of generalization: Proceedings: Sfi-cnls workshop on formal approaches to supervised learning (1992: Santa fe, nm), 1995.

Naftali Tishby, Esther Levin, and Sara A Solla. Consistent inference of probabilities in layered networks: Predictions and generalization. In *International Joint Conference on Neural Networks*, volume 2, pages 403–409, 1989.

Esther Levin, Naftali Tishby, and Sara A Solla. A statistical approach to learning and generalization in layered neural networks. *Proceedings of the IEEE*, 78(10):1568–1574, 1990.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992a.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minimum search finds simple nets. Technical report, 1994. Technical Report FKI-200-94.

Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems*, pages 529–536, 1995.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997a.

John Shawe-Taylor and Robert C Williamson. A pac analysis of a bayesian estimator. In *Annual Workshop on Computational Learning Theory: Proceedings of the tenth annual conference on Computational learning theory*, volume 6, pages 2–9, 1997.

John Shawe-Taylor, Peter L Bartlett, Robert C Williamson, and Martin Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE transactions on Information Theory*, 44(5):1926–1940, 1998.

- David A McAllester. Some pac-bayesian theorems. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 230–234. ACM, 1998.
- Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016. URL <http://arxiv.org/abs/1609.04836>.
- Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5949–5958, 2017.
- Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.
- Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*, 2017. URL <http://auai.org/uai2017/proceedings/papers/173.pdf>.
- Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the imagenet scale: a pac-bayesian compression approach. 2018.
- Radford M Neal. Priors for infinite networks (tech. rep. no. crg-tr-94-1). *University of Toronto*, 1994.
- Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.
- Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. URL <https://arxiv.org/abs/1611.01232>.

Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018.

Adri Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow gaussian processes. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=Bklfsi0cKm>.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2019a. URL <https://openreview.net/forum?id=B1g30j0qF7>.

Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019a.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

Harris Drucker, Christopher JC Burges, Linda Kaufman, Alex J Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161, 1997.

Dan C Cireşan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.
- Harry A Pierson and Michael S Gashler. Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31(16):821–835, 2017.
- Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 19(6):1236–1246, 2018.
- Georgios N Yannakakis and Julian Togelius. *Artificial intelligence and games*, volume 2. Springer, 2018.
- Dan Guest, Kyle Cranmer, and Daniel Whiteson. Deep learning and its application to lhc physics. *Annual Review of Nuclear and Particle Science*, 68:161–181, 2018.
- David Foster. *Generative deep learning: teaching machines to paint, write, compose, and play*. O’Reilly Media, 2019.
- Mihaela Robila and Stefan A Robila. Applications of artificial intelligence methodologies to behavioral and social sciences. *Journal of Child and Family Studies*, pages 1–13, 2019.
- Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017a. URL <https://arxiv.org/abs/1611.03530>.
- Andrei Nikolaevich Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information and Transmission*, 1(1):3–11, 1965.

Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.

Alexander K Zvonkin and Leonid A Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25(6):83, 1970.

LI Ming and Paul MB Vitányi. Kolmogorov complexity and its applications. *Algorithms and Complexity*, 1:187, 2014.

Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

Tor Lattimore and Marcus Hutter. No free lunch versus occams razor in supervised learning. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, pages 223–235. Springer, 2013.

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Occam’s razor. *Information processing letters*, 24(6):377–380, 1987.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Peter Gács. *Lecture notes on descriptional complexity and randomness*. Citeseer, 1988.

Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature communications*, 9(1):761, 2018a.

Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *CoRR*, abs/1710.05468, 2017. URL <http://arxiv.org/abs/1710.05468>.

Tomaso Poggio, Kenji Kawaguchi, Qianli Liao, Brando Miranda, Lorenzo Rosasco, Xavier Boix, Jack Hidary, and Hrushikesh Mhaskar. Theory of deep learning iii: the non-overfitting puzzle. Technical report, CBMM memo 073, 2018.

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*, 2019.

Yiding Jiang, Pierre Foret, Scott Yak, Daniel M. Roy, Hossein Mobahi, Gintare Karolina Dziugaite, Samy Bengio, Suriya Gunasekar, Isabelle Guyon, and Behnam Neyshabur. Neurips 2020 competition: Predicting generalization in deep learning, 2020.

Guillermo Valle-Pérez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *arXiv preprint arXiv:1805.08522*, 2018.

Chris Mingard, Guillermo Valle-Pérez, Joar Skalse, and Ard A Louis. Is sgd a bayesian sampler? well, almost. *arXiv preprint arXiv:2006.15191*, 2020.

Guillermo Valle-Perez and Ard A. Louis. Generalization theory of deep learning.

Chris Mingard, Joar Skalse, Guillermo Valle-Pérez, David Martínez-Rubio, Vladimir Mikulik, and Ard A Louis. Neural networks are a priori biased towards boolean functions with low entropy. *arXiv preprint arXiv:1909.11522*, 2019.

Jürgen Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

Christian Van den Broeck and Ryoichi Kawai. Learning in feedforward boolean networks. *Physical Review A*, 42(10):6210, 1990.

P Carnevali and S Patarnello. Exhaustive thermodynamical analysis of boolean learning networks. *EPL (Europhysics Letters)*, 4(10):1199, 1987.

Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature Communications*, 9(1):1–7, 2018b.

Leonardo Franco. Generalization ability of boolean functions implemented in feedforward neural networks. *Neurocomputing*, 70(1):351–361, 2006.

Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight VC-dimension bounds for piecewise linear neural networks. In *Proceedings of the 2017 Conference on Learning Theory*, volume 65 of *Proceedings of Machine Learning Research*, pages 1064–1068. PMLR, 07–10 Jul 2017. URL <http://proceedings.mlr.press/v65/harvey17a.html>.

Eric B Baum and David Haussler. What size net gives valid generalization? In *Advances in neural information processing systems*, pages 81–90, 1989.

Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on information theory*, 22(1):75–81, 1976.

Leonardo Franco and Martin Anthony. On a generalization complexity measure for boolean functions. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 973–978. IEEE, 2004.

Ehud Friedgut. Boolean functions with low average sensitivity depend on few coordinates. *Combinatorica*, 18(1):27–35, 1998.

Sam F Greenbury, Steffen Schaper, Sebastian E Ahnert, and Ard A Louis. Genetic correlations greatly increase mutational robustness and can both reduce and enhance evolvability. *PLoS computational biology*, 12(3):e1004773, 2016.

David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S. Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, Aaron Courville, Simon Lacoste-Julien, and Yoshua Bengio. A closer look at memorization in deep networks. *Proceedings of the 34th International Conference on Machine Learning (ICML’17)*, 2017a. URL <https://arxiv.org/abs/1706.05394>.

David H Wolpert and R Waters. The relationship between pac, the statistical physics framework, the bayesian framework, and the vc framework. In *In. Citeseer*, 1994.

E Estevez-Rams, R Lora Serrano, B Aragón Fernández, and I Brito Reyes. On the non-randomness of maximum lempel ziv complexity sequences of finite size. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 23(2):023118, 2013.

Steffen Schaper and Ard A Louis. The arrival of the frequent: how bias in genotype-phenotype maps can steer populations to local optima. *PLoS one*, 9(2):e86635, 2014.

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Giacomo De Palma, Bobak T Kiani, and Seth Lloyd. Random deep neural networks are biased towards simple functions: supplementary material.

Greg Yang and Hadi Salman. A fine-grained spectral perspective on neural networks. *arXiv preprint arXiv:1907.10599*, 2019.

Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature Communications*, 9(1):761, 2018c.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368, 2016.

Bhiksha Raj. Neural networks: What can a network represent, 2018. URL <http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Spring.2018/www/slides/lec2.universal.pdf>.

Martin Anthony. *Discrete mathematics of neural networks: selected topics*, volume 8. Siam, 2001.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=B1EA-M-0Z>.

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow gaussian processes. *arXiv preprint arXiv:1808.05587*, 2018.

Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian convolutional neural networks with many channels are gaussian processes. *arXiv preprint arXiv:1810.05148*, 2018a.

Kyle Luther and H Sebastian Seung. Variance-preserving initialization schemes improve deep network training: But which variance is preserved? *arXiv preprint arXiv:1902.04942*, 2019.

David Page. How to train your resnet 7: Batch norm, 2019. URL <https://myrtle.ai/how-to-train-your-resnet-7-batch-norm/>.

Peter L Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. 1996.

Steve Lawrence, C. Lee Giles, and Ah Chung Tsoi. What size neural network gives optimal generalization? convergence properties of backpropagation. Technical report, 1996.

Vaishnavh Nagarajan and J Zico Kolter. Uniform convergence may be unable to explain generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 11611–11622, 2019.

Gintare Karolina Dziugaite, Alexandre Drouin, Brady Neal, Nitarshan Rajkumar, Ethan Caballero, Linbo Wang, Ioannis Mitliagkas, and Daniel M Roy. In search of robust measures of generalization. *arXiv preprint arXiv:2010.11924*, 2020.

David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*, 2020.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

- Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A Alemi, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. 2019b. URL <https://openreview.net/forum?id=Sk1D9yrFPS>.
- Jonathan S Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales. *arXiv preprint arXiv:1909.12673*, 2019.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- Stefano Spigler, Mario Geiger, and Matthieu Wyart. Asymptotic learning curves of kernel methods: empirical data vs teacher-student paradigm. *arXiv preprint arXiv:1905.10843*, 2019.
- Peter Sollich. Gaussian process regression with mismatched models. In *Advances in Neural Information Processing Systems*, pages 519–526, 2002.
- Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.
- Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019.
- Benjamin Guedj. A primer on pac-bayesian learning. *arXiv preprint arXiv:1901.05353*, 2019.
- Omar Rivasplata, Ilja Kuzborskij, Csaba Szepesvári, and John Shawe-Taylor. Pac-bayes analysis beyond the usual bounds. *arXiv preprint arXiv:2006.13057*, 2020.
- Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum dependent learning curves in kernel regression and wide neural networks. *arXiv preprint arXiv:2002.02561*, 2020.

Jeffrey Negrea, Gintare Karolina Dziugaite, and Daniel M Roy. In defense of uniform convergence: Generalization via derandomization with an application to interpolating predictors. *arXiv preprint arXiv:1912.04265*, 2019.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=BygfhAcYX>.

Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

Vladimir Koltchinskii. *Oracle Inequalities in Empirical Risk Minimization and Sparse Recovery Problems: Ecole d’Eté de Probabilités de Saint-Flour XXXVIII-2008*, volume 2033. Springer Science & Business Media, 2011.

Benjamin Guedj John Shawe-Taylor. A primer on pac-bayesian learning. 2019.

Andreas Maurer. A note on the pac bayesian theorem. *arXiv preprint cs/0411099*, 2004.

John Langford and John Shawe-Taylor. Pac-bayes & margins. In *Advances in neural information processing systems*, pages 439–446, 2003.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Peter L Bartlett. For valid generalization the size of the weights is more important than the size of the network. In *Advances in neural information processing systems*, pages 134–140, 1997.

Peter L Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536, 1998.

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*, 2018a. URL https://openreview.net/forum?id=Skz_WfbCZ.

Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. In *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 297–299. PMLR, 06–09 Jul 2018. URL <http://proceedings.mlr.press/v75/golowich18a.html>.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018b.

Andrew R Barron and Jason M Klusowski. Complexity, statistical risk, and metric entropy of deep nets using total path variation. *arXiv preprint arXiv:1902.00800*, 2019.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401, 2015a.

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 254–263. PMLR, 10–15 Jul 2018. URL <http://proceedings.mlr.press/v80/arora18b.html>.

Wesley J Maddox, Gregory Benton, and Andrew Gordon Wilson. Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint arXiv:2003.02139*, 2020.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in neural information processing systems*, pages 8572–8583, 2019.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis

of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019a.

Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In *Advances in Neural Information Processing Systems*, pages 10836–10846, 2019.

Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT ’93, pages 5–13, New York, NY, USA, 1993. ACM. ISBN 0-89791-611-5. doi: 10.1145/168304.168306. URL <http://doi.acm.org/10.1145/168304.168306>.

Yao Zhang, Andrew M Saxe, Madhu S Advani, and Alpha A Lee. Energy–entropy competition and the effectiveness of stochastic gradient descent in machine learning. *Molecular Physics*, 116(21–22):1–10, 2018.

Lei Wu, Zhanxing Zhu, et al. Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv preprint arXiv:1706.10239*, 2017.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Vaishnavh Nagarajan and Zico Kolter. Deterministic pac-bayesian generalization bounds for deep networks via generalizing noise-resilience. 2018.

Arindam Banerjee, Tiancong Chen, and Yingxue Zhou. De-randomized pac-bayes margin bounds: Applications to non-convex and non-smooth predictors. *arXiv preprint arXiv:2002.09956*, 2020.

Gintare Karolina Dziugaite and Daniel M Roy. Data-dependent pac-bayes priors via differential privacy. In *Advances in Neural Information Processing Systems*, pages 8430–8441, 2018.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

- Nick Littlestone and Manfred Warmuth. Relating data compression and learnability. 1986.
- Alon Brutzkus, Amir Globerson, Eran Malach, and Shai Shalev-Shwartz. SGD learns over-parameterized networks that provably generalize on linearly separable data. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=rJ33wwxRb>.
- Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of machine learning research*, 2(Mar):499–526, 2002.
- Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1225–1234. PMLR, 20–22 Jun 2016. URL <http://proceedings.mlr.press/v48/hardt16.html>.
- Wenlong Mou, Liwei Wang, Xiyu Zhai, and Kai Zheng. Generalization bounds of sgld for non-convex learning: Two theoretical viewpoints. In *Conference On Learning Theory*, pages 605–638, 2018.
- Ilja Kuzborskij and Christoph H Lampert. Data-dependent stability of stochastic gradient descent. *arXiv preprint arXiv:1703.01678*, 2017.
- Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Learnability, stability and uniform convergence. *Journal of Machine Learning Research*, 11(Oct):2635–2670, 2010.
- Vitaly Feldman and Jan Vondrák. High probability generalization bounds for uniformly stable algorithms with nearly optimal rate. *arXiv preprint arXiv:1902.10710*, 2019.
- Ben London. A pac-bayesian analysis of randomized learning with application to stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 2931–2940, 2017.
- Jian Li, Xuanyuan Luo, and Mingda Qiao. On generalization error bounds of noisy gradient methods for non-convex learning. *arXiv preprint arXiv:1902.00621*, 2019.

Yi Zhou, Yingbin Liang, and Huishuai Zhang. Understanding generalization error of sgd in nonconvex optimization. *stat*, 1050:7, 2019a.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

Zachary Charles and Dimitris Papailiopoulos. Stability and generalization of learning algorithms that converge to global optima. In *International Conference on Machine Learning*, pages 744–753, 2018.

Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. 2018.

Colin Wei and Tengyu Ma. Data-dependent sample complexity of deep neural networks via lipschitz augmentation. *arXiv preprint arXiv:1905.03684*, 2019a.

Colin Wei and Tengyu Ma. Improved sample complexities for deep networks and robust classification via an all-layer margin. *arXiv preprint arXiv:1910.04284*, 2019b.

Omry Cohen, Or Malka, and Zohar Ringel. Learning curves for deep neural networks: A gaussian field theory perspective. *CoRR*, abs/1906.05301, 2019a. URL <http://arxiv.org/abs/1906.05301>.

John Langford and Matthias Seeger. Bounds for averaging classifiers. 2001.

Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. *CoRR*, abs/1710.06451, 2017. URL <http://arxiv.org/abs/1710.06451>.

Pascal Germain, Francis Bach, Alexandre Lacoste, and Simon Lacoste-Julien. Pac-bayesian theory meets bayesian inference. In *Advances in Neural Information Processing Systems*, pages 1884–1892, 2016.

Adrià Garriga-Alonso, Laurence Aitchison, and Carl Edward Rasmussen. Deep convolutional networks as shallow Gaussian processes. *arXiv preprint arXiv:1808.05587*, August 2018. URL <https://arxiv.org/abs/1808.05587>.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling, 2020.

Greg Yang. Tensor programs i: Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *arXiv preprint arXiv:1910.12478*, pages 9951–9960, 2019b.

Stanislaw Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos J Storkey. Finding flatter minima with sgd. In *ICLR (Workshop)*, 2018.

Mingwei Wei and David J Schwab. How noise affects the hessian spectrum in overparameterized neural networks. *arXiv preprint arXiv:1910.00195*, 2019.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International Conference on Machine Learning*, pages 343–351, 2013.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proc. 3rd Int. Conf. Learn. Representations*, 2014.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *Advances in neural information processing systems*, pages 630–637, 1990.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings*

of the 32nd International Conference on Machine Learning, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015a. PMLR. URL <http://proceedings.mlr.press/v37/ioffe15.html>.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *ICLR (Workshop)*, 2015b.

Chiyuan Zhang, Qianli Liao, Alexander Rakhlin, Karthik Sridharan, Brando Miranda, Noah Golowich, and Tomaso Poggio. Musings on deep learning: Properties of sgd. 04/2017 2017b. URL <https://cbmm.mit.edu/publications/musings-deep-learning-properties-sgd>. formerly titled "Theory of Deep Learning III: Generalization Properties of SGD".

Tomaso Poggio, Qianli Liao, and Andrzej Banburski. Complexity control by gradient descent in deep networks. *Nature communications*, 11(1):1–5, 2020.

Omry Cohen, Or Malka, and Zohar Ringel. Learning curves for deep neural networks: a gaussian field theory perspective. *arXiv preprint arXiv:1906.05301*, 2019b.

Florian Wenzel, Kevin Roth, Bastiaan S Veeling, Jakub Świątkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*, 2020.

Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 18(134):1–35, 2017.

Alexander G de G Matthews, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Sample-then-optimize posterior sampling for bayesian linear models. In *NIPS Workshop on Advances in Approximate Bayesian Inference*, 2017.

Alexander G. de G. Matthews, Jiri Hron, Mark Rowland, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=H1-nGgWC->.

- John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.
- Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.
- Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
- Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don’t decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.
- David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, and Aaron Courville. Deep nets don’t learn via memorization. *International Conference on Learning Representations Workshop*, 2017b. URL <https://openreview.net/forum?id=HJC2SzZCW>.
- Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL <http://arxiv.org/abs/1708.07747>.
- Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997b.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Kamaludin Dingle, Guillermo Valle Pérez, and Ard A Louis. Generic predictions of output probability based on complexities of inputs and outputs. *Scientific Reports*, 10(1):1–9, 2020.

David Wales et al. *Energy landscapes: Applications to clusters, biomolecules and glasses*. Cambridge University Press, 2003.

Steven H Strogatz. *Nonlinear dynamics and chaos with student solutions manual: With applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

Claire P Massen and Jonathan PK Doye. Power-law distributions for the areas of the basins of attraction on a potential energy landscape. *Physical Review E*, 75(3):037101, 2007.

Andrew J Ballard, Ritankar Das, Stefano Martiniani, Dhagash Mehta, Levent Sagun, Jacob D Stevenson, and David J Wales. Energy landscapes for machine learning. *Physical Chemistry Chemical Physics*, 19(20):12585–12603, 2017.

Gadi Naveh, Oded Ben-David, Haim Sompolinsky, and Zohar Ringel. Predicting the outputs of finite networks trained with noisy gradients. *arXiv preprint arXiv:2004.01190*, 2020.

Nicolas Brosse, Alain Durmus, and Eric Moulines. The promises and pitfalls of stochastic gradient langevin dynamics. In *Advances in Neural Information Processing Systems*, pages 8268–8278, 2018.

Kamaludin Dingle, Steffen Schaper, and Ard A Louis. The structure of the genotype–phenotype map strongly constrains the evolution of non-coding rna. *Interface focus*, 5(6):20150053, 2015.

David L Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture*, 1(2000):32, 2000.

Alon Brutzkus, Amir Globerson, Eran Malach, and Shai Shalev-Shwartz. Sgd learns over-parameterized networks that provably generalize on linearly separable data. *arXiv preprint arXiv:1710.10174*, 2017.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1019–1028. JMLR.org, PMLR, 06–11 Aug 2017. URL <http://proceedings.mlr.press/v70/dinh17b.html>.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. *arXiv preprint arXiv:1806.08734*, 2018.

Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. *arXiv preprint arXiv:1706.05394*, 2017.

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*, 2018b. URL <https://openreview.net/forum?id=HJC2SzZCW>.

Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. Non-vacuous generalization bounds at the imagenet scale: a PAC-bayesian compression approach. In *International Conference on Learning Representations*, 2019b. URL <https://openreview.net/forum?id=BJgqqsAct7>.

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in neural information processing systems*, pages 6158–6169, 2019.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204, 2015.

Marco Baity-Jesi, Levent Sagun, Mario Geiger, Stefano Spigler, Gérard Ben Arous, Chiara Cammarota, Yann LeCun, Matthieu Wyart, and Giulio Biroli. Comparing dynamics: Deep neural networks versus glassy systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124013, 2019.

Simon Becker, Yao Zhang, et al. Geometry of energy landscapes and the optimizability of deep neural networks. *Physical Review Letters*, 124(10):108301, 2020.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019.

Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

M. Li and P.M.B. Vitanyi. *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag New York Inc, 2008.

L.A. Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, 10(3):30–35, 1974.

Giacomo De Palma, Bobak Toussi Kiani, and Seth Lloyd. Random deep neural networks are biased towards simple functions. *arXiv preprint arXiv:1812.10156*, 2018.

Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

David JC Mackay. Introduction to gaussian processes. *NATO ASI series. Series F: computer and system sciences*, pages 133–165, 1998.

- Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8139–8148, 2019b.
- David A McAllester. Pac-bayesian model averaging. In *COLT*, volume 99, pages 164–170. Citeseer, 1999.
- Elliott Sober. *Ockham’s razors*. Cambridge University Press, 2015.
- David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992b.
- Carl Edward Rasmussen and Zoubin Ghahramani. Occam’s razor. In *Advances in neural information processing systems*, pages 294–300, 2001.
- Pedro Domingos. The role of occam’s razor in knowledge discovery. *Data mining and knowledge discovery*, 3(4):409–425, 1999.
- Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6):1076–1136, 2011.
- Tom F Sterkenburg. Solomonoff prediction and occam’s razor. *Philosophy of Science*, 83(4):459–479, 2016.
- Matthias Hein and Jean-Yves Audibert. Intrinsic dimensionality estimation of submanifolds in rd. In *Proceedings of the 22nd international conference on Machine learning*, pages 289–296, 2005.
- Sebastian Goldt, Marc Mézard, Florent Krzakala, and Lenka Zdeborová. Modelling the influence of data structure on learning in neural networks. *arXiv preprint arXiv:1909.11500*, 2019.
- Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015b.
- Edwin Fong and CC Holmes. On the marginal likelihood and cross-validation. *Biometrika*, 107(2):489–496, 2020.

- Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8789–8798, 2018.
- Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL <http://yann.lecun.com/exdb/mnist/>.
- Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.
- Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.
- Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL <http://www.cs.toronto.edu/~kriz/cifar.html>.
- Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. URL <https://arxiv.org/abs/1409.1556>.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b.
- Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.
- Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

- Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.
- Vincent G Sigillito, Simon P Wing, Larrie V Hutton, and Kile B Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10(3):262–266, 1989.
- Greg Yang and Sam S Schoenholz. Deep mean field theory: Layerwise variance and width variation as methods to control gradient explosion. 2018.
- Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.
- Madhu S Advani and Andrew M Saxe. High-dimensional dynamics of generalization error in neural networks. *arXiv preprint arXiv:1710.03667*, 2017.
- Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1309–1318. PMLR, 10–15 Jul 2018. URL <http://proceedings.mlr.press/v80/draxler18a.html>.