Towards an Integrated Understanding of Neural Networks

by

David Rolnick

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mathematics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2018

Signature redacted

Author .....
Department of Mathematics
August 10, 2018

Signature redacted

Certified by .....
Nir Shavit
Professor
Thesis Supervisor

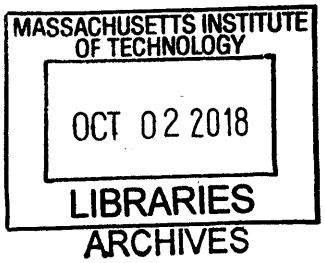
Signature redacted

Certified by .....
Edward S. Boyden
Professor
Thesis Supervisor

Certified by .....
Max Tegmark
Professor
Thesis Supervisor

Signature redacted

Accepted by .....
Jonathan Kelner
Chairman, Department Committee on Graduate Theses



Towards an Integrated Understanding of Neural Networks

by

David Rolnick

Submitted to the Department of Mathematics
on August 10, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Mathematics

Abstract

Neural networks underpin both biological intelligence and modern AI systems, yet there is relatively little theory for how the observed behavior of these networks arises. Even the connectivity of neurons within the brain remains largely unknown, and popular deep learning algorithms lack theoretical justification or reliability guarantees. This thesis aims towards a more rigorous understanding of neural networks. We characterize and, where possible, prove essential properties of neural algorithms: expressivity, learning, and robustness. We show how observed emergent behavior can arise from network dynamics, and we develop algorithms for learning more about the network structure of the brain.

Thesis Supervisor: Nir Shavit
Title: Professor

Thesis Supervisor: Edward S. Boyden
Title: Professor

Thesis Supervisor: Max Tegmark
Title: Professor

Acknowledgments

I am indebted to my thesis committee, Nir Shavit, Edward S. Boyden, Max Tegmark, Ankur Moitra, and Jonathan Kelner, for their advice, support, and mentorship through my Ph.D.; to my qualifying examiners, Jacob Fox and Henry Cohn; to my collaborators and coauthors (in alphabetical order), Kevin Aydin, Serge Belongie, Jeremy Bernstein, David Budden, Carina Curto, Ishita Dasgupta, Jesús A. De Loera, Roozbeh Farhoodi, Noah Golowich, Boris Hanin, Viren Jain, Thouis Raymond Jones, Shahab Kamali, Seymour Knowles-Barley, Konrad Kording, Reuben N. La Haye, Jeff William Lichtman, Henry Lin, Susan Margulies, Alexander Matveev, Yaron Meirovitch, Vahab Mirrokni, Richard A. Moy, Amir Najmi, Gergely Odor, Toufiq Parag, Michael Pernpeintner, Hanspeter Pfister, Eric Riedl, Hayk Saribekyan, Pablo Soberón, Haim Sompolinsky, Gwen Spencer, Despina Stasi, Jon Swenson, Andreas Veit, and Praveen Venkataramana; and to the many other people who offered advice on this research, including Scott Aaronson, Hannah Alpert, Noah Arbesfeld, Alexander Barvinok, David Cox, Tom Dean, Joseph A. Gallian, Surya Ganguli, Samuel Gershman, Adam Hesterberg, Daphne Ippolito, Tanya Khovanova, Daniel Kriz, Peter Li, Ricky Liu, Jeremy Maitin-Shepard, Adam Marblestone, Pablo Parrilo, Tomaso Poggio, Art Pope, Krishanu Sankar, Agnes Szanto, Y. William Yu, Ramakrishna Vedantam, and many anonymous referees. Finally, I am immensely grateful to all of my family and friends for their continued inspiration and encouragement.

I would like to thank the MIT Mathematics Department, the Computer Science and Artificial Intelligence Laboratory (CSAIL), the MIT Media Lab, Google Research, and the Center for Brains, Minds, & Machines for their support and stimulating working environments. I am grateful for the NSF Graduate Research Fellowship Program (NSF grant no. 1122374) and the MIT Hugh Hampton Young Memorial Fund Fellowship. Work included in this thesis was also supported by the Foundational Questions Institute, the Rothberg Family Fund for Cognitive Science, the AOL Connected Experiences Laboratory, a Google Focused Research Award, a Google Faculty Research Award, AWS Cloud Credits for Research, and a Facebook equipment donation.

Contents

Chapter 1

Introduction

Neural networks are the foundation of both biological intelligence and modern AI systems. Within the brain, nerve cells, or neurons, send and receive electrical signals. Networks of neurons are the physical seat of sensation, memory, and thought. Artificial neural networks, based loosely on their biological analogues, are mathematical constructions in which simple functions receive input and output from each other in a manner similar to nerve cells, leading to complex emergent behavior. Such artificial networks are the basis of deep learning algorithms enabling computers to learn from data, and are now used widely in applications ranging from self-driving cars to automated medical diagnoses.

Increasingly, we are able to observe our own brains in action and to design surprisingly powerful AI algorithms, but our understanding of the underlying neural networks is shallow. When they work, we cannot adequately explain why. When they fail, we often do not know what is going wrong or how to fix it.

Within the biological brain, new scientific tools are opening up a wealth of data on the structure of neural circuits and their electrical firing patterns; but unifying interpretations for this data are elusive. Even at the scale of individual neurons, there is debate about exactly how signal integration occurs between inputs. The behavior of neuron ensembles and the algorithms by which these ensembles learn remain conjectural. Emergent properties of entire networks, such as rational thought, are completely beyond our current understanding. Humanity is impatient for cures

to Alzheimer's disease and other neurological conditions, yet explaining even normal brain function remains beyond our grasp.

Likewise, as society becomes increasingly reliant upon AI, it is vital to understand our own creations from a rigorous scientific and mathematical perspective. Industry research in AI is largely focused on results, not fundamentals. This has led to incremental improvements on prior work, rather than innovation built from the ground up. Our lack of knowledge also raises the possibility of unexpected behavior or even catastrophic failure. AI must have performance guarantees if it is to be trusted as a major force within society.

The fields of artificial intelligence and neuroscience have much to gain from one another via a unified perspective on neural networks. Early developments in AI, such as convolutional neural networks, were originally motivated by the brain; but current work in deep learning has drifted away from neuroscience. Many researchers pay lip service to "biologically plausible" algorithms, but these generally have little to do with actual biology. Nevertheless, some of the toughest problems in AI today, such as how to perform credit assignment in reinforcement learning, may have already been solved in biological neural networks. Reciprocally, artificial intelligence can help neuroscientists understand the brain better; since it is far easier to experiment with a computer than with a living organism.

In order to advance an integrated understanding of neural networks, I believe that each of the following steps is essential.

1. **Mathematical proofs** characterizing the essential properties of neural networks responsible for fundamental principles of success or failure (such as the ability to express many functions of the input, or the ability to learn an individual function from example data). Such mathematical analysis is necessarily dependent on the dynamics of the networks in question but is independent of the implementation details and also is substrate-agnostic between computers and brains.

2. **Scientific experiments** on artificial networks where fully theoretical analysis

is inapplicable or challenging. In this way, we can treat machine learning algorithms similarly to unknown biological systems, rather than merely evaluating their performance against benchmarks.

3. **Data acquisition** at scale on the organization of biological systems, preparatory to a similar conceptual analysis of their algorithms as with artificial systems. At a minimum, it appears critical to know the network structure and the nature of the signals transmitted between (or within) neurons.

4. **Statistical analysis** of the structural and functional primitives that best explain the data for biological networks. Subtleties of individual circuits, while important biologically, may ultimately be inessential to understanding the overall algorithmic organization of a brain.

5. **Mathematical models** based on these structural and functional primitives. Characteristics of biological networks can ideally be matched to those of artificial networks - either existing network models, wholly new models, or some combination of the two.

These proposed research directions may be compared to traditional approaches both in deep learning and in computational neuroscience.

On the deep learning side, a scientific and mathematical approach complements the engineering focus found both in industry and in much academic research. Evaluation of algorithms is traditionally limited to measuring their performance on a number of well-known datasets. Evaluation on these datasets does not give us actionable insights into what is causing improvement or degradation in performance. Ultimately, however, the effectiveness of deep learning depends on general properties such as expressivity, learnability, and robustness. Some of these properties, it turns out, can be provably tired to algorithm design, while in other cases they can be empirically evaluated in isolated. Note that such an empirical approach to modes of algorithmic success and failure is very different from simply calculating the performance on a benchmark dataset.

It may appear strange to run experiments on networks that are completely defined by specified sets of equations. After all, we know exactly what the networks will do in given circumstances, because they do exactly what we tell them to do. The goal, however, is not to recover these equations; it is to identify principles of emergent behavior, much as the results of statistical physics derive ultimately from the Schrödinger equation of quantum mechanics but are not immediately obvious from it. Furthermore, the artificial nature of the hardware involved in deep learning allows a far greater ability to record and modify the behavior of an artificial network than in a biological brain, where state-of-the-art tools from fMRI to calcium imaging to optogenetics often suffer from a range of limitations, notably (1) poor spatial or temporal resolution and (2) difficulty in working with more than small sets of neurons in a single experiment.

On the computational neuroscience side, many questions remain which may possibly be addressed by statistical and mathematical analysis of the large datasets that increasingly are becoming available. For example: How does the structure of individual neurons and of networks arise during development? What loss functions and learning rules are used as the network learns? Do subnetworks arise that are specialized in algorithmic primitives such as memory storage, statistical inference, or symbolic manipulation?

Research has historically often focused on the patterns associated with particular behaviors (e.g. navigation, facial recognition, birdsong, etc.) in well-defined regions of the brain. While such approaches have yielded important neuroscientific and medical knowledge, they may not be well-suited to an understanding of neural network algorithms. Often those tasks and brain regions best suited to study in isolation are precisely those that are organized in idiosyncratic ways independent from the rest of cognition. Narrow focus makes data acquisition and interpretation easier, but is not necessarily reflective of the broad principles by which networks are organized structurally or how they operate and learn. More fundamentally, a focus on correlating neuronal activity with the task being studied neglects the cause of such dynamics, which from an algorithmic point of view is much more important than exactly which

neurons are active when.

In this thesis, I will present progress towards each of the five steps in the above outline for an integrated understanding of artificial and biological neural networks. Not included in this thesis is various work on combinatorics [127, 101, 120], graph theory [119, 44], and discrete algorithms [33, 141, 121, 123, 124, 31, 32], also completed during my graduate studies.

1.1 Expressivity and Learnability

The power of artificial, as well as biological, neural networks ultimately comes from their ability effectively to capture complicated functions to approximate patterns observed in input data. This ability is actually two abilities: First, there exist neural networks that can approximate a given function of the input data. Second, in many case these networks can be learned from the data using methods such as backpropagation. Neither property would be useful without the other - a perfect approximation that cannot be effectively learned is little better than the input data itself, and likewise for a poor approximation that can be learned. While both expressibility and learnability have been observed in deep learning systems, there is relatively little theoretical justification for the circumstances in which functions can be expressed and learned by neural networks.

For the problem of expressibility (Chapter 2), we provide formal justification for the observed advantage of *deeper* learning. Even neural networks with a single hidden layer are universal approximators, but deep networks tend in practice to be more powerful than shallow ones. Several recent breakthroughs in deep learning, such as *residual networks* (ResNets) [53], are effective because they permit the training of deeper networks than was previously feasible. Recurrent networks may be thought of as a limiting case representing arbitrarily large depth. We prove that the total number of neurons m required to approximate natural classes of multivariate polynomials of n variables grows only linearly with n for deep neural networks, but grows exponentially when merely a single hidden layer is allowed. We also provide evidence that when

the number of hidden layers is increased from 1 to k, the neuron requirement grows exponentially not with n but with $n^{1/k}$, suggesting that the minimum number of layers required for practical expressibility grows only logarithmically with n.

For the problem of learnability (Chapter 3), we investigate the effects of initialization and architecture on the start of training in deep ReLU nets. We identify two common failure modes for early training in which the mean and variance of activations are poorly behaved. For each failure mode, we give a rigorous proof of when it occurs at initialization and how to avoid it. The first failure mode, exploding/vanishing mean activation length, can be avoided by initializing weights from a symmetric distribution with variance 2/fan-in. The second failure mode, exponentially large variance of activation length, can be avoided by keeping constant the sum of the reciprocals of layer widths. We demonstrate empirically the effectiveness of our theoretical results in predicting when networks are able to start training. In particular, we note that many popular initializations fail our criteria, whereas correct initialization and architecture allows much deeper networks to be trained.

These chapters are based on Rolnick and Tegmark [125] and Lin, Tegmark, and Rolnick [84], in which I formulated the proofs on expressibility and ran empirical experiments to support them, and on Hanin and Rolnick [50], in which I contributed to proofs and ran empirical experiments. Also see work in Benjamin, Rolnick, and Kording [13].

1.2 Robustness

Beyond expressivity and learnability, another property is vital for neural networks to learn effectively from real-world data. In practice, training data is subject to error both in the input examples and in the desired outputs. This is an inevitable consequence of training on ever larger datasets; even those that are heavily curated are not error-free. Moreover, well-annotated datasets can be time-consuming and expensive to collect, lending increased interest to larger but noisy datasets that are more easily obtained, such as data pulled automatically from the Internet. One

reason that deep neural networks have been successful is that they can learn from such imperfect training data, unlike algorithms such as the perceptron learning rule.

In Chapter 4, we investigate the behavior of deep neural networks on training sets with massively noisy labels. We demonstrate remarkably high test performance after training on corrupted data from MNIST, CIFAR, and ImageNet. For example, on MNIST we obtain test accuracy above 90 percent even after each clean training example has been diluted with 100 randomly-labeled examples. Such behavior holds across multiple patterns of label noise, even when erroneous labels are biased towards confusing classes. We show that training in this regime requires a significant but manageable increase in dataset size that is related to the factor by which correct labels have been diluted. Finally, we provide an analysis of our results that shows how increasing noise decreases the effective batch size.

This chapter is based on Rolnick, Veit, Belongie, and Shavit [126], in which I designed the project, ran experiments, and interpreted them.

1.3 Connectomics algorithms

The connectivity of a biological neural network may be represented as a graph, called the *connectome*. In the case of a human brain, the connectome contains roughly 10^{11} vertices (neurons) and 10^{15} edges (synapses). Understanding information flow through a brain likely requires some knowledge of the underlying connectome. Currently, however, the connectome has not been mapped for any organism except the simple worm *C. elegans*, for which there are only 300 vertices in the graph. Even describing such a small connectome took several years of human labor. Accordingly, there is much interest in algorithms for reconstructing the connectome automatically from high-resolution microscope images. This data is quite challenging, and, to date, all such connectomics algorithms have required extensive manual correction, which is itself unscalable to networks of reasonable size.

In Chapter 6, we consider how to automate the error-correction process in connectomics. Typical deep learning algorithms rely upon localized classification of in-

dividual pixels, rather than overall morphology. This leads to a high incidence of erroneously merged objects. Humans, by contrast, can easily detect such errors by acquiring intuition for the correct morphology of objects. Biological neurons have complicated and variable shapes, which are challenging to learn, and merge errors take a multitude of different forms. We present an algorithm, MergeNet, that shows 3D ConvNets can, in fact, detect merge errors from high-level neuronal morphology. MergeNet follows unsupervised training and operates across datasets. We demonstrate the performance of MergeNet both on a variety of connectomics data and on a dataset created from merged MNIST images.

This chapter is based on Rolnick et al. [122], in which I helped design the algorithm, implemented it, and tested it. Also see work in Meirovitch et al. [94].

1.4 Neuronal branching

Information flow within the brain remains mysterious even at the scale of individual neurons. There is debate, for example, about exactly how signal integration occurs between inputs to the same neuron. The functional properties of neurons are intimately tied to their intricate branching structures, which are observed to differ between neuron types. Neither the development nor the function of these morphologies has been adequately explained, despite the increasing amounts of data available for rigorous analysis.

In Chapter 5, we present statistical models that explain much of the variance in branching morphologies demonstrated by neurons, incorporating distance from the soma and asymmetric structure by using Markovian hidden states. By analyzing the models, we observe that many databases of neuron morphologies follow a similar pattern; symmetric branching close to the soma and asymmetric branching for distant parts. This is witnessed by the fact that the frequency of asymmetrical motifs, as captured by graph theoretical trees, is higher than symmetrical motifs. Statistical models of branching in neuron morphologies promise to make their analysis more meaningful and suggest directions for investigating underlying biological processes.

This chapter is based on Farhoodi, Rolnick, and Kording [39] (further work in preparation), in which I helped design and implement the statistical analyses.

1.5 Attractor networks

It is generally supposed that specific structures within the brain contribute to particular kinds of computation; however, what these structures are, and what algorithmic primitives they correspond to, remains in doubt. Possibly, for example, the brain includes neural subnetworks particularly well-suited to memory storage, statistical inference, or symbolic manipulation. It has been suggested that some such subnetworks may be hard-coded into genetic instructions for building the brain, while others may develop through interactions with the environment [90]. How such emergent properties arise from ensembles of individual neurons remains a mystery, both in understanding computation in the brain and in building effective artificial networks with similar behavior. In Chapter 7, we consider two models that replicate behavior observed in the brain using attractor networks (networks in which the dynamics fall into stable *attractor* states).

First, we consider the problem of stochastic computation. Stochasticity is an essential part of explaining the world. Increasingly, neuroscientists and cognitive scientists are identifying mechanisms whereby the brain uses probabilistic reasoning in representational, predictive, and generative settings. But stochasticity is not always useful: robust perception and memory retrieval require representations that are immune to corruption by stochastic noise. In an effort to combine these robust representations with stochastic computation, we present a Hopfield-type network that generalizes traditional recurrent attractor networks to follow probabilistic Markov dynamics between stable and noise-resistant fixed points.

Next, we model periodic attractor states in which sequences of neurons are cyclically active, as has been observed in the cortex and hippocampus of the brain. We rigorously prove the existence of such states within simple threshold-linear networks, providing the first justification of the complex attractors observed in these networks.

This chapter is based on Rolnick, Bernstein, Dasgupta, and Sompolinsky (further work in preparation), in which I helped design, implement, and test the network, and on work in preparation with Carina Curto, in which I formulated the proofs.

Chapter 2

The Power of Deeper Networks for Expressing Natural Functions

2.1 Introduction

Deep learning has lately been shown to be a very powerful tool for a wide range of problems, from image segmentation to machine translation. Despite its success, many of the techniques developed by practitioners of artificial neural networks (ANNs) are heuristics without theoretical guarantees. Perhaps most notably, the power of feedforward networks with many layers (*deep* networks) has not been fully explained. The goal of this chapter is to shed more light on this question and to suggest heuristics for how deep is deep enough.

It is well-known [29, 40, 60, 7, 112] that neural networks with a single hidden layer can approximate any function under reasonable assumptions, but it is possible that the networks required will be extremely large. Recent authors have shown that some functions can be approximated by deeper networks much more efficiently (i.e. with fewer neurons) than by shallower ones. Often, these results admit one or more of the following limitations: "existence proofs" without explicit constructions of the functions in question; explicit constructions, but relatively complicated functions; or applicability only to types of network rarely used in practice.

It is important and timely to extend this work to make it more concrete and

actionable, by deriving resource requirements for approximating natural classes of functions using today's most common neural network architectures. Lin, Tegmark, and Rolnick [84] recently proved that it is exponentially more efficient to use a deep network than a shallow network when Taylor-approximating the product of input variables. In the present work, we move far beyond this result in the following ways: (i) we use standard uniform approximation instead of Taylor approximation, (ii) we show that the exponential advantage of depth extends to all general sparse multivariate polynomials, and (iii) we address the question of how the number of neurons scales with the number of layers. Our results apply to standard feedforward neural networks and are borne out by empirical tests.

Our primary contributions are as follows:

- It is possible to achieve **arbitrarily close approximations** of simple multivariate and univariate polynomials with neural networks having a **bounded number of neurons** (see §2.2).
- Such polynomials are **exponentially easier to approximate with deep networks than with shallow networks** (see §2.3).
- The power of networks improves rapidly with depth; for natural polynomials, **the number of layers required is at most logarithmic** in the number of input variables, where the base of the logarithm depends upon the layer width (see §2.4).

2.1.1 Related Work

Deeper networks have been shown to have greater representational power with respect to various notions of complexity, including piecewise linear decision boundaries [98] and topological invariants [14]. Recently, Poole et al. [116] and Raghu et al. [117] showed that the trajectories of input variables attain exponentially greater length and curvature with greater network depth.

Work including [30, 37, 112, 115, 150] shows that there exist functions that require exponential width to be approximated by a shallow network. Barron [7] pro-

vides bounds on the error in approximating general functions by shallow networks. Mhaskar, Liao, and Poggio [95] and Poggio et al. [115] show that for *compositional functions* (those that can be expressed by recursive function composition), the number of neurons required for approximation by a deep network is exponentially smaller than the best known upper bounds for a shallow network. Mhaskar et al. [95] ask whether functions with tight lower bounds must be pathologically complicated, a question which we answer here in the negative.

Various authors have also considered the power of deeper networks of types other than the standard feedforward model. The problem has also been posed for sum-product networks [34] and restricted Boltzmann machines [91]. Cohen, Sharir, and Shashua [24] showed, using tools from tensor decomposition, that shallow arithmetic circuits can express only a measure-zero set of the functions expressible by deep circuits. A weak generalization of this result to convolutional neural networks was shown by Cohen and Shashua [25].

2.2 The power of approximation

In this chapter, we will consider the standard model of feedforward neural networks (also called *multilayer perceptrons*). Formally, the network may be considered as a multivariate function $N(\mathbf{x}) = \mathbf{A}_k\sigma(\cdots\sigma(\mathbf{A}_1\sigma(\mathbf{A}_0\mathbf{x}))\cdots)$, where $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_k$ are constant matrices and σ denotes a scalar nonlinear function applied element-wise to vectors. The constant k is referred to as the *depth* of the network. The *neurons* of the network are the entries of the vectors $\sigma(\mathbf{A}_\ell\cdots\sigma(\mathbf{A}_1\sigma(\mathbf{A}_0\mathbf{x}))\cdots)$, for $\ell = 1, \ldots, k-1$. These vectors are referred to as the *hidden layers* of the network.

Two notions of approximation will be relevant in our results: *ϵ-approximation*, also known as *uniform approximation*, and *Taylor approximation*.

Definition 2.2.1. *For constant $\epsilon > 0$, we say that a network $N(\mathbf{x})$* ϵ-approximates *a multivariate function $f(\mathbf{x})$ (for $\mathbf{x}$ in a specified domain $(-R, R)^n$) if $\sup_{\mathbf{x}} |N(\mathbf{x}) - f(\mathbf{x})| < \epsilon$.*

Definition 2.2.2. *We say that a network $N(\mathbf{x})$* Taylor-approximates *a multivariate*

polynomial $p(\mathbf{x})$ of degree d if $p(\mathbf{x})$ is the dth order Taylor polynomial (about the origin) of $N(\mathbf{x})$.

The following proposition shows that Taylor approximation implies ϵ-approximation for homogeneous polynomials. The reverse implication does not hold.

Proposition 2.2.3. *Suppose that the network $N(\mathbf{x})$ Taylor-approximates the homogeneous multivariate polynomial $p(\mathbf{x})$. Then, for every ϵ, there exists a network $N_\epsilon(\mathbf{x})$ that ϵ-approximates $p(\mathbf{x})$, such that $N(\mathbf{x})$ and $N_\epsilon(\mathbf{x})$ have the same number of neurons in each layer. (This statement holds for $\mathbf{x} \in (-R, R)^n$ for any specified R.)*

Proof. Suppose that $N(\mathbf{x}) = \mathbf{A}_k\sigma(\cdots\sigma(\mathbf{A}_1\sigma(\mathbf{A}_0\mathbf{x}))\cdots)$ and that $p(\mathbf{x})$ has degree d. Since $p(\mathbf{x})$ is a Taylor approximation of $N(\mathbf{x})$, we can write $N(\mathbf{x})$ as $p(\mathbf{x}) + E(\mathbf{x})$, where $E(\mathbf{x}) = \sum_{i=d+1}^{\infty} E_i(\mathbf{x})$ is a Taylor series with each $E_i(\mathbf{x})$ homogeneous of degree i. Since $N(\mathbf{x})$ is the function defined by a neural network, it converges for every $\mathbf{x} \in \mathbb{R}^n$. Thus, $E(\mathbf{x})$ converges, as does $E(\delta\mathbf{x})/\delta^d = \sum_{i=d+1}^{\infty} \delta^{i-d}E_i(\mathbf{x})$. By picking δ sufficiently small, we can make each term $\delta^{i-d}E_i(\mathbf{x})$ arbitrarily small. Let δ be small enough that $|E(\delta\mathbf{x})/\delta^d| < \epsilon$ holds for all $\mathbf{x}$ in $(-R, R)^n$.

Let $\mathbf{A}'_0 = \delta\mathbf{A}_0$, $\mathbf{A}'_k = \mathbf{A}_k/\delta^d$, and $\mathbf{A}'_\ell = \mathbf{A}_\ell$ for $\ell = 1, 2, \ldots, k-1$. Then, for $N_\epsilon(\mathbf{x}) = \mathbf{A}'_k\sigma(\cdots\sigma(\mathbf{A}'_1\sigma(\mathbf{A}'_0\mathbf{x}))\cdots)$, we observe that $N_\epsilon(\mathbf{x}) = N(\delta\mathbf{x})/\delta^d$, and therefore:

$$\begin{aligned} |N_\epsilon(\mathbf{x}) - p(\mathbf{x})| &= |N(\delta\mathbf{x})/\delta^d - p(\mathbf{x})| \\ &= |p(\delta\mathbf{x})/\delta^d + E(\delta\mathbf{x})/\delta^d - p(\mathbf{x})| \\ &= |E(\delta\mathbf{x})/\delta^d| \\ &< \epsilon. \end{aligned}$$

We conclude that $N_\epsilon(\mathbf{x})$ is an ϵ-approximation of $p(\mathbf{x})$, as desired. □

For a fixed nonlinear function σ, we consider the total number of neurons (excluding input and output neurons) needed for a network to approximate a given function. Remarkably, it is possible to attain arbitrarily good approximations of a (not necessarily homogeneous) multivariate polynomial by a feedforward neural network, even

with a single hidden layer, without increasing the number of neurons past a certain bound. (See also Corollary 1 in [115].)

Theorem 2.2.4. *Suppose that $p(\mathbf{x})$ is a degree-d multivariate polynomial and that the nonlinearity σ has nonzero Taylor coefficients up to degree d. Let $m_k^\epsilon(p)$ be the minimum number of neurons in a depth-k network that ϵ-approximates p. Then, the limit $\lim_{\epsilon\to 0} m_k^\epsilon(p)$ exists (and is finite). (Once again, this statement holds for $\mathbf{x} \in (-R, R)^n$ for any specified R.)*

Proof. We show that $\lim_{\epsilon\to 0} m_1^\epsilon(p)$ exists; it follows immediately that $\lim_{\epsilon\to 0} m_k^\epsilon(p)$ exists for every k, since an ϵ-approximation to p with depth k can be constructed from one with depth 1.

Let $p_1(\mathbf{x}), p_2(\mathbf{x}), \ldots, p_s(\mathbf{x})$ be the monomials of $p(\mathbf{x})$, so that $p(\mathbf{x}) = \sum_i p_i(\mathbf{x})$. We claim that each $p_i(\mathbf{x})$ can be Taylor-approximated by a network $N^i(\mathbf{x})$ with one hidden layer. This follows, for example, from the proof in [84] that products can be Taylor-approximated by networks with one hidden layer, since each monomial is the product of several inputs (with multiplicity); we prove a far stronger result about $N^i(\mathbf{x})$ later in this work (see Theorem 2.3.1).

Suppose now that $N^i(\mathbf{x})$ has m_i hidden neurons. By Proposition 2.2.3, we conclude that since $p_i(\mathbf{x})$ is homogeneous, it may be δ-approximated by a network $N_\delta^i(\mathbf{x})$ with m_i hidden neurons, where $\delta = \epsilon/s$. By combining the networks $N_\delta^i(\mathbf{x})$ for each i, we can define a network $N_\epsilon(\mathbf{x}) = \sum_i N_\delta^i(\mathbf{x})$ with $\sum_i m_i$ neurons. Then, we have:

$$\begin{aligned} |N_\epsilon(\mathbf{x}) - p(\mathbf{x})| &\leq \sum_i |N_\delta^i(\mathbf{x}) - p_i(\mathbf{x})| \\ &\leq \sum_i \delta = s\delta = \epsilon. \end{aligned}$$

Hence, $N_\epsilon(\mathbf{x})$ is an ϵ-approximation of $p(\mathbf{x})$, implying that $m_1^\epsilon(p) \leq \sum_i m_i$ for every ϵ. Thus, $\lim_{\epsilon\to 0} m_1^\epsilon(p)$ exists, as desired. □

This theorem is perhaps surprising, since it is common for ϵ-approximations to functions to require ever-greater complexity, approaching infinity as $\epsilon \to 0$. For example, the function $\exp(|-x|)$ may be approximated on the domain $(-\pi, \pi)$ by Fourier

sums of the form $\sum_{k=0}^{m} a_m \cos(kx)$. However, in order to achieve ϵ-approximation, we need to take $m \sim 1/\sqrt{\epsilon}$ terms. By contrast, we have shown that a finite neural network architecture can achieve arbitrarily good approximations merely by altering its weights.

Note also that the assumption of nonzero Taylor coefficients cannot be dropped from Theorem 2.2.4. For example, the theorem is false for rectified linear units (ReLUs), which are piecewise linear and do not admit a Taylor series. This is because ϵ-approximating a non-linear polynomial with a piecewise linear function requires an ever-increasing number of pieces as $\epsilon \to 0$.

Theorem 2.2.4 allows us to make the following definition:

Definition 2.2.5. *Suppose that a nonlinear function σ is given. For p a multivariate polynomial, let $m_k^{uniform}(p)$ be the minimum number of neurons in a depth-k network that ϵ-approximates p for all ϵ arbitrarily small. Set $m^{uniform}(p) = \min_k m_k^{uniform}(p)$. Likewise, let $m_k^{Taylor}(p)$ be the minimum number of neurons in a depth-k network that Taylor-approximates p, and set $m^{Taylor}(p) = \min_k m_k^{Taylor}(p)$.*

In the next section, we will show that there is an exponential gap between $m_1^{\text{uniform}}(p)$ and $m^{\text{uniform}}(p)$ and between $m_1^{\text{Taylor}}(p)$ and $m^{\text{Taylor}}(p)$ for various classes of polynomials p.

2.3 The inefficiency of shallow networks

In this section, we compare the efficiency of shallow networks (those with a single hidden layer) and deep networks at approximating multivariate polynomials. Proofs of our main results are included in §2.6.

2.3.1 Multivariate polynomials

Our first result shows that uniform approximation of monomials requires exponentially more neurons in a shallow than a deep network.

Theorem 2.3.1. *Let $p(\mathbf{x})$ denote the monomial $x_1^{r_1}x_2^{r_2}\cdots x_n^{r_n}$, with $d = \sum_{i=1}^n r_i$. Suppose that the nonlinearity σ has nonzero Taylor coefficients up to degree $2d$. Then, we have:*

(i) $m_1^{uniform}(p) = \prod_{i=1}^n (r_i + 1)$,

(ii) $m^{uniform}(p) \leq \sum_{i=1}^n (7\lceil \log_2(r_i) \rceil + 4)$,

where $\lceil x \rceil$ denotes the smallest integer that is at least x.

We can prove a comparable result for m^{Taylor} under slightly weaker assumptions on σ. Note that by setting $r_1 = r_2 = \ldots = r_n = 1$, we recover the result of [84] that the product of n numbers requires 2^n neurons in a shallow nctwork but can be Taylor-approximated with linearly many neurons in a deep network.

Theorem 2.3.2. *Let $p(\mathbf{x})$ denote the monomial $x_1^{r_1}x_2^{r_2}\cdots x_n^{r_n}$, with $d = \sum_{i=1}^n r_i$. Suppose that σ has nonzero Taylor coefficients up to degree d. Then, we have:*

(i) $m_1^{Taylor}(p) = \prod_{i=1}^n (r_i + 1)$,

(ii) $m^{Taylor}(p) \leq \sum_{i=1}^n (7\lceil \log_2(r_i) \rceil + 4)$.

It is worth noting that neither of Theorems 2.3.1 and 2.3.2 implies the other. This is because it is possible for a polynomial to admit a compact uniform approximation without admitting a compact Taylor approximation.

It is natural now to consider the cost of approximating general polynomials. However, without further constraint, this is relatively uninstructive because polynomials of degree d in n variables live within a space of dimension $\binom{n+d}{d}$, and therefore most require exponentially many neurons for *any* depth of network. We therefore consider polynomials of *sparsity* c: that is, those that can be represented as the sum of c monomials. This includes many natural functions.

The following theorem, when combined with Theorems 2.3.1 and 2.3.2, shows that general polynomials p with subexponential sparsity have exponentially large $m_1^{\text{uniform}}(p)$ and $m_1^{\text{Taylor}}(p)$, but subexponential $m^{\text{uniform}}(p)$ and $m^{\text{Taylor}}(p)$.

Theorem 2.3.3. *Let $p(\mathbf{x})$ be a multivariate polynomial of degree d and sparsity c, having monomials $q_1(\mathbf{x}), q_2(\mathbf{x}), \ldots, q_c(\mathbf{x})$. Suppose that the nonlinearity σ has nonzero Taylor coefficients up to degree $2d$. Then, we have:*

(i) $m_1^{uniform}(p) \geq \frac{1}{c} \max_j m_1^{uniform}(q_j)$.

(ii) $m^{uniform}(p) \leq \sum_j m^{uniform}(q_j)$.

These statements also hold if $m^{uniform}$ is replaced with m^{Taylor}.

As mentioned above with respect to ReLUs, some assumptions on the Taylor coefficients of the activation function are necessary for the results we present. However, it is possible to loosen the assumptions of Theorem 2.3.1 and 2.3.2 while still obtaining exponential lower bounds on $m_1^{\text{uniform}}(p)$ and $m_1^{\text{Taylor}}(p)$:

Theorem 2.3.4. *Let $p(\mathbf{x})$ denote the monomial $x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}$, with $d = \sum_{i=1}^n r_i$. Suppose that the nonlinearity σ has nonzero dth Taylor coefficient (other Taylor coefficients are allowed to be zero). Then, $m_1^{uniform}(p)$ and $m_1^{Taylor}(p)$ are at least $\frac{1}{d} \prod_{i=1}^n (r_i + 1)$. (An even better lower bound is the maximum coefficient in the polynomial $\prod_i (1 + y + \ldots + y^{r_i})$.)*

2.3.2 Univariate polynomials

As with multivariate polynomials, depth can offer an exponential savings when approximating univariate polynomials. We show below (Proposition 2.3.5) that a shallow network can approximate any degree-d univariate polynomial with a number of neurons at most linear in d. The monomial x^d requires $d + 1$ neurons in a shallow network (Proposition 2.3.6), but can be approximated with only logarithmically many neurons in a deep network. Thus, depth allows us to reduce networks from linear to logarithmic size, while for multivariate polynomials the gap was between exponential and linear. The difference here arises because the dimensionality of the space of univariate degree-d polynomials is linear in d, which the dimensionality of the space of multivariate degree-d polynomials is exponential in d.

Proposition 2.3.5. *Suppose that the nonlinearity σ has nonzero Taylor coefficients up to degree d. Then, $m_1^{Taylor}(p) \leq d+1$ for every univariate polynomial p of degree d.*

Proof. Pick $a_0, a_1, \ldots, a_d$ to be arbitrary, distinct real numbers. Consider the Vandermonde matrix $\mathbf{A}$ with entries $A_{ij} = a_i^j$. It is well-known that $\det(\mathbf{A}) = \prod_{i<i'}(a_{i'} - a_i) \neq 0$. Hence, $\mathbf{A}$ is invertible, which means that multiplying its columns by nonzero values gives another invertible matrix. Suppose that we multiply the jth column of $\mathbf{A}$ by σ_j to get $\mathbf{A}'$, where $\sigma(x) = \sum_j \sigma_j x^j$ is the Taylor expansion of $\sigma(x)$.

Now, observe that the ith row of $\mathbf{A}'$ is exactly the coefficients of $\sigma(a_i x)$, up to the degree-d term. Since $\mathbf{A}'$ is invertible, the rows must be linearly independent, so the polynomials $\sigma(a_i x)$, restricted to terms of degree at most d, must themselves be linearly independent. Since the space of degree-d univariate polynomials is $(d+1)$-dimensional, these $d+1$ linearly independent polynomials must span the space. Hence, $m_1^{\text{Taylor}}(p) \leq d+1$ for any univariate degree-d polynomial p. In fact, we can fix the weights from the input neuron to the hidden layer (to be $a_0, a_1, \ldots, a_d$, respectively) and still represent any polynomial p with $d+1$ hidden neurons. □

Proposition 2.3.6. *Let $p(x) = x^d$, and suppose that the nonlinearity $\sigma(x)$ has nonzero Taylor coefficients up to degree $2d$. Then, we have:*

(i) $m_1^{uniform}(p) = d+1$.

(ii) $m^{uniform}(p) \leq 7\lceil \log_2(d) \rceil$.

These statements also hold if $m^{uniform}$ is replaced with m^{Taylor}.

Proof. Part (i) follows from part (i) of Theorems 2.3.1 and 2.3.2 by setting $n = 1$ and $r_1 = d$.

For part (ii), observe that we can Taylor-approximate the square x^2 of an input x with three neurons in a single layer:

$$\frac{1}{2\sigma''(0)}\left(\sigma(x) + \sigma(-x) - 2\sigma(0)\right) = x^2 + \mathcal{O}(x^4 + x^5 + \ldots).$$

We refer to this construction as a *square gate*, and the construction of Lin et al. [84] as a *product gate*. We also use *identity gate* to refer to a neuron that simply preserves the input of a neuron from the preceding layer (this is equivalent to the *skip connections* in residual nets [53]).

Consider a network in which each layer contains a square gate (3 neurons) and either a product gate or an identity gate (4 or 1 neurons, respectively), according to the following construction: The square gate squares the output of the preceding square gate, yielding inductively a result of the form x^{2^k}, where k is the depth of the layer. Writing d in binary, we use a product gate if there is a 1 in the 2^{k-1}-place; if so, the product gate multiplies the output of the preceding product gate by the output of the preceding square gate. If there is a 0 in the 2^{k-1}-place, we use an identity gate instead of a product gate. Thus, each layer computes x^{2^k} and multiplies $x^{2^{k-1}}$ to the computation if the 2^{k-1}-place in d is 1. The process stops when the product gate outputs x^d.

This network clearly uses at most $7\lceil \log_2(d) \rceil$ neurons, with a worst case scenario where $d+1$ is a power of 2. Hence $m^{\text{Taylor}}(p) \leq 7\lceil \log_2(d) \rceil$, with $m^{\text{uniform}}(p) \leq m^{\text{Taylor}}(p)$ by Proposition 2.2.3 since p is homogeneous. □

2.4 How efficiency improves with depth

We now consider how $m_k^{\text{uniform}}(p)$ scales with k, interpolating between exponential in n (for $k = 1$) and linear in n (for $k = \log n$). In practice, networks with modest $k > 1$ are effective at representing natural functions. We explain this theoretically by showing that the cost of approximating the product polynomial drops off rapidly as k increases.

By repeated application of the shallow network construction in [84], we obtain the following upper bound on $m_k^{\text{uniform}}(p)$, which we conjecture to be essentially tight. Our approach leverages the compositionality of polynomials, as discussed e.g. in [95] and [115], using a tree-like neural network architecture.

Theorem 2.4.1. *Let $p(\mathbf{x})$ equal the product $x_1 x_2 \cdots x_n$, and suppose σ has nonzero*

Taylor coefficients up to degree n. Then, we have:

$$m_k^{uniform}(p) = \mathcal{O}\left(n^{(k-1)/k} \cdot 2^{n^{1/k}}\right). \tag{2.1}$$

Proof. We construct a network in which groups of the n inputs are recursively multiplied up to Taylor approximation. The n inputs are first divided into groups of size b_1, and each group is multiplied in the first hidden layer using 2^{b_1} neurons (as described in [84]). Thus, the first hidden layer includes a total of $2^{b_1} n/b_1$ neurons. This gives us n/b_1 values to multiply, which are in turn divided into groups of size b_2. Each group is multiplied in the second hidden layer using 2^{b_2} neurons. Thus, the second hidden layer includes a total of $2^{b_2} n/(b_1 b_2)$ neurons.

We continue in this fashion for $b_1, b_2, \ldots, b_k$ such that $b_1 b_2 \cdots b_k = n$, giving us one neuron which is the product of all of our inputs. By considering the total number of neurons used, we conclude

$$m_k^{\text{Taylor}}(p) \leq \sum_{i=1}^{k} \frac{n}{\prod_{j=1}^{i} b_j} 2^{b_i} = \sum_{i=1}^{k} \left(\prod_{j=i+1}^{k} b_j \right) 2^{b_i}. \tag{2.2}$$

By Proposition 2.2.3, $m_k^{\text{uniform}}(p) \leq m_k^{\text{Taylor}}(p)$ since p is homogeneous. Setting $b_i = n^{1/k}$, for each i, gives us the desired bound (2.1). □

In fact, we can solve for the choice of b_i such that the upper bound in (2.2) is minimized, under the condition $b_1 b_2 \cdots b_k = n$. Using the technique of Lagrange multipliers, we know that the optimum occurs at a minimum of the function

$$\mathcal{L}(b_i, \lambda) := \left(n - \prod_{i=1}^{k} b_i \right) \lambda + \sum_{i=1}^{k} \left(\prod_{j=i+1}^{k} b_j \right) 2^{b_i}.$$

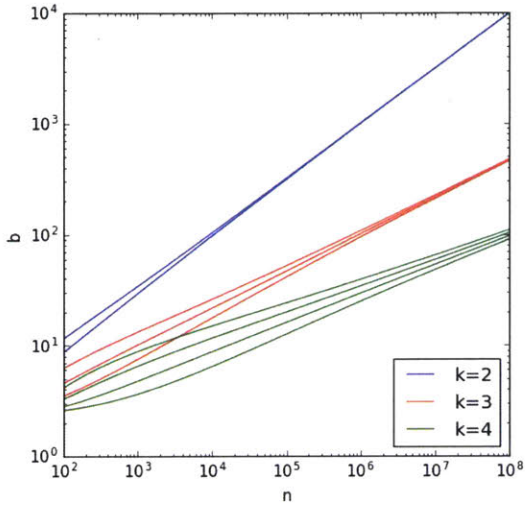


Figure 2-1: The optimal settings for $\{b_i\}_{i=1}^{k}$ as n varies are shown for $k = 1, 2, 3$. Observe that the b_i converge to $n^{1/k}$ for large n, as witnessed by a linear fit in the log-log plot. The exact values are given by equations (2.4) and (2.5).

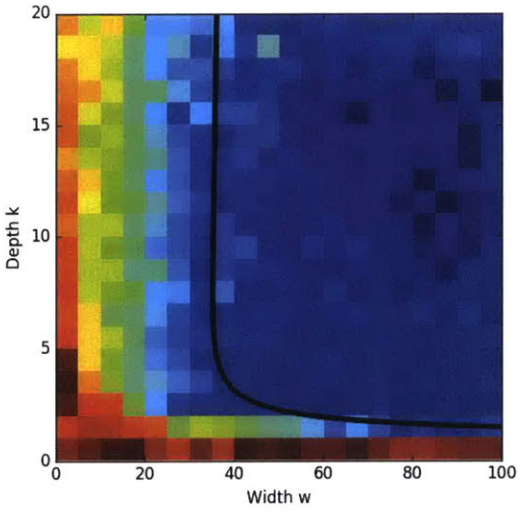


Figure 2-2: Performance of trained networks in approximating the product of 20 input variables, ranging from red (high error) to blue (low error). The error shown here is the expected absolute difference between the predicted and actual product. The curve $w = n^{(k-1)/k} \cdot 2^{n^{1/k}}$ for $n = 20$ is shown in black. In the region above and to the right of the curve, it is possible to effectively approximate the product function (Theorem 2.4.1).

Differentiating $\mathcal{L}$ with respect to b_i, we obtain the conditions

$$0 = -\lambda \prod_{j \neq i} b_j + \sum_{h=1}^{i-1} \left(\frac{\prod_{j=h+1}^{k} b_j}{b_i} \right) 2^{b_h} + (\log 2) \left(\prod_{j=i+1}^{k} b_j \right) 2^{b_i}, \text{ for } 1 \leq i \leq k \quad (2.3)$$

$$0 = n - \prod_{j=1}^{k} b_j. \quad (2.4)$$

Dividing (2.3) by $\prod_{j=i+1}^{k} b_j$ and rearranging gives us the recursion

$$b_i = b_{i-1} + \log_2(b_{i-1} - 1/\log 2). \quad (2.5)$$

Thus, the optimal b_i are not exactly equal but very slowly increasing with i (see Figure 2-1).

The following conjecture states that the bound given in Theorem 2.4.1 is (approximately) optimal.

Conjecture 2.4.2. *Let $p(\mathbf{x})$ equal to the product $x_1 x_2 \cdots x_n$, and suppose that σ has all nonzero Taylor coefficients. Then, we have:*

$$m_k^{uniform}(p) = 2^{\Theta(n^{1/k})}, \quad (2.6)$$

i.e., the exponent grows as $n^{1/k}$ for $n \to \infty$.

We empirically tested Conjecture 2.4.2 by training ANNs to predict the product of input values $x_1, \ldots, x_n$ with $n = 20$ (see Figure 2-2). The rapid interpolation from exponential to linear width aligns with our predictions.

In our experiments, we used feedforward networks with dense connections between successive layers. In the figure, we show results for $\sigma(x) = \tanh(x)$ (note that this behavior is even better than expected, since this function actually has numerous zero Taylor coefficients). Similar results were also obtained for rectified linear units (ReLUs) as the nonlinearity, despite the fact that this function does not even admit a Taylor series. The number of layers was varied, as was the number of neurons

within a single layer. The networks were trained using the AdaDelta optimizer [160] to minimize the absolute value of the difference between the predicted and actual values. Input variables x_i were drawn uniformly at random from the interval $[0, 2]$, so that the expected value of the output would be of manageable size.

Eq. (2.6) provides a helpful rule of thumb for how deep is deep enough. Suppose, for instance, that we wish to keep typical layers no wider than about a thousand ($\sim 2^{10}$) neurons. Eq. (2.6) then implies $n^{1/k} \lesssim 10$, *i.e.*, that the number of layers should be at least

$$k \gtrsim \log_{10} n.$$

It would be very interesting if one could show that general polynomials p in n variables require a superpolynomial number of neurons to approximate for any constant number of hidden layers. The analogous statement for Boolean circuits - whether the complexity classes TC^0 and TC^1 are equal - remains unresolved and is assumed to be quite hard. Note that the formulations for Boolean circuits and deep neural networks are independent statements (neither would imply the other) due to the differences between computation on binary and real values. Indeed, gaps in expressivity have already been proven to exist for real-valued neural networks of different depths, for which the analogous results remain unknown in Boolean circuits (see e.g. [96, 23, 22, 98, 24, 150]).

2.5 Conclusion

We have shown how the power of deeper ANNs can be quantified even for simple polynomials. We have proved that arbitrarily good approximations of polynomials are possible even with a fixed number of neurons and that there is an exponential gap between the width of shallow and deep networks required for approximating a given sparse polynomial. For n variables, a shallow network requires size exponential in n, while a deep network requires at most linearly many neurons. Networks with a constant number $k > 1$ of hidden layers appear to interpolate between these extremes, following a curve exponential in $n^{1/k}$. This suggests a rough heuristic for the number of

layers required for approximating simple functions with neural networks. For example, if we want no layers to have more than 2^{10} neurons, say, then the minimum number of layers required grows only as $\log_{10} n$. To further improve efficiency using the $\mathcal{O}(n)$ constructions we have presented, it suffices to increase the number of layers by a factor of $\log_2 10 \approx 3$, to $\log_2 n$.

The key property we use in our constructions is compositionality, as detailed in [115]. It is worth noting that as a consequence our networks enjoy the property of *locality* mentioned in [24], which is also a feature of convolutional neural nets. That is, each neuron in a layer is assumed to be connected only to a small subset of neurons from the previous layer, rather than the entirety (or some large fraction). In fact, we showed (e.g. Prop. 2.3.6) that there exist natural functions computable with linearly many neurons, with each neuron is connected to at most *two* neurons in the preceding layer, which nonetheless cannot be computed with fewer than exponentially many neurons in a single layer, no matter how may connections are used. Our construction can also be framed with reference to the other properties mentioned in [24]: those of *sharing* (in which weights are shared between neural connections) and *pooling* (in which layers are gradually collapsed, as our construction essentially does with recursive combination of inputs).

This section has focused exclusively on the resources (neurons and synapses) required to *compute* a given function for fixed network depth. (Note also results of [86, 51, 48] for networks of fixed width.) An important complementary challenge is to quantify the resources (e.g. training steps) required to *learn* the computation, i.e., to converge to appropriate weights using training data — possibly a fixed amount thereof, as suggested in Zhang et al. [161]. There are simple functions that can be computed with polynomial resources but require exponential resources to learn [137]. It is quite possible that architectures we have not considered increase the feasibility of learning. For example, residual networks (ResNets) [53] and unitary nets (see e.g. [3, 66]) are no more powerful in representational ability than conventional networks of the same size, but by being less susceptible to the "vanishing/exploding gradient" problem, it is far easier to optimize them in practice. We look forward to

future work that will help us understand the power of neural networks to learn.

2.6 Proofs of Theorems

Proof of Theorem 2.3.1.

Without loss of generality, suppose that $r_i > 0$ for $i = 1, \ldots, n$. Let X be the multiset in which x_i occurs with multiplicity r_i.

We first show that $\prod_{i=1}^{n}(r_i + 1)$ neurons are *sufficient* to approximate $p(\mathbf{x})$. Appendix A in [84] demonstrates that for variables $y_1, \ldots, y_N$, the product $y_1 \cdot \cdots \cdot y_N$ can be Taylor-approximated as a linear combination of the 2^N functions $\sigma(\pm y_1 \pm \cdots \pm y_d)$.

Consider setting $y_1, \ldots, y_d$ equal to the elements of multiset X. Then, we conclude that we can approximate $p(\mathbf{x})$ as a linear combination of the functions $\sigma(\pm y_1 \pm \cdots \pm y_d)$. However, these functions are not all distinct: there are $r_i + 1$ distinct ways to assign $\pm$ signs to r_i copies of x_i (ignoring permutations of the signs). Therefore, there are $\prod_{i=1}^{n}(r_i+1)$ distinct functions $\sigma(\pm y_1 \pm \cdots \pm y_N)$, proving that $m^{\text{Taylor}}(p) \leq \prod_{i=1}^{n}(r_i+1)$. Proposition 2.2.3 implies that for homogeneous polynomials p, we have $m_1^{\text{uniform}}(p) \leq m_1^{\text{Taylor}}(p)$.

We now show that this number of neurons is also *necessary* for approximating $p(\mathbf{x})$. Suppose that $N_\epsilon(\mathbf{x})$ is an ϵ-approximation to $p(\mathbf{x})$ with depth 1, and let the Taylor series of $N_\epsilon(\mathbf{x})$ be $p(\mathbf{x}) + E(\mathbf{x})$. Let $E_k(\mathbf{x})$ be the degree-k homogeneous component of $E(\mathbf{x})$, for $0 \leq k \leq 2d$. By the definition of ϵ-approximation, $\sup_{\mathbf{x}} E(\mathbf{x})$ goes to 0 as ϵ does, so by picking ϵ small enough, we can ensure that the coefficients of each $E_k(\mathbf{x})$ go to 0.

Let $m = m_1^{\text{uniform}}(p)$ and suppose that $\sigma(x)$ has the Taylor expansion $\sum_{k=0}^{\infty} \sigma_k x^k$. Then, by grouping terms of each order, we conclude that there exist constants a_{ij} and

w_j such that

$$\sigma_d \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^d = p(\mathbf{x}) + E_d(\mathbf{x})$$
$$\sigma_k \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^k = E_k(\mathbf{x}) \quad \text{for } k \neq d.$$

For each $S \subseteq X$, let us take the derivative of this equation by every variable that occurs in S, where we take multiple derivatives of variables that occur multiple times. This gives

$$\frac{\sigma_d \cdot d!}{|S|!} \sum_{j=1}^{m} w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{d-|S|} = \frac{\partial}{\partial S} p(\mathbf{x}) + \frac{\partial}{\partial S} E_d(\mathbf{x}), \tag{2.7}$$
$$\frac{\sigma_k \cdot k!}{|S|!} \sum_{j=1}^{m} w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{k-|S|} = \frac{\partial}{\partial S} E_k(\mathbf{x}). \tag{2.8}$$

Observe that there are $r \equiv \prod_{i=1}^{n} (r_i + 1)$ choices for S, since each variable x_i can be included anywhere from 0 to r_i times. Define $\mathbf{A}$ to be the $r \times m$ matrix with entries $A_{S,j} = \prod_{h \in S} a_{hj}$. We claim that $\mathbf{A}$ has full row rank. This would show that the number of columns m is at least the number of rows $r = \prod_{i=1}^{n} (r_i + 1)$, proving the desired lower bound on m.

Suppose towards contradiction that the rows $A_{S_\ell, \bullet}$ admit a linear dependence:

$$\sum_{\ell=1}^{r} c_\ell A_{S_\ell, \bullet} = \mathbf{0},$$

where the coefficients c_ℓ are all nonzero and the S_ℓ denote distinct subsets of X. Let S_* be such that $|c_*|$ is maximized. Then, take the dot product of each side of the above

equation by the vector with entries (indexed by j) equal to $w_j \left(\sum_{i=1}^n a_{ij}x_i\right)^{d-|S_*|}$:

$$\begin{aligned}
0 &= \sum_{\ell=1}^{r} c_\ell \sum_{j=1}^{m} w_j \prod_{h\in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij}x_i\right)^{d-|S_*|} \\
&= \sum_{\ell|(|S_\ell|=|S_*|)} c_\ell \sum_{j=1}^{m} w_j \prod_{h\in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij}x_i\right)^{d-|S_\ell|} \\
&+ \sum_{\ell|(|S_\ell|\neq|S_*|)} c_\ell \sum_{j=1}^{m} w_j \prod_{h\in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij}x_i\right)^{(d+|S_\ell|-|S_*|)-|S_\ell|}.
\end{aligned}$$

We can use (2.7) to simplify the first term and (2.8) (with $k = d + |S_\ell| - |S_*|$) to simplify the second term, giving us:

$$\begin{aligned}
0 &= \sum_{\ell|(|S_\ell|=|S_*|)} c_\ell \cdot \frac{|S_\ell|!}{\sigma_d \cdot d!} \cdot \left(\frac{\partial}{\partial S_\ell} p(\mathbf{x}) + \frac{\partial}{\partial S_\ell} E_d(\mathbf{x})\right) \qquad (2.9) \\
&+ \sum_{\ell|(|S_\ell|\neq|S_*|)} c_\ell \cdot \frac{|S_\ell|!}{\sigma_{d+|S_\ell|-|S_*|} \cdot (d+|S_\ell|-|S_*|)!} \cdot \frac{\partial}{\partial S_\ell} E_{d+|S_\ell|-|S_*|}(\mathbf{x})
\end{aligned}$$

Consider the coefficient of the monomial $\frac{\partial}{\partial S_*} p(\mathbf{x})$, which appears in the first summand with coefficient $c_* \cdot \frac{|S_*|!}{\sigma_d \cdot d!}$. Since the S_ℓ are distinct, this monomial does not appear in any other term $\frac{\partial}{\partial S_\ell} p(\mathbf{x})$, but it could appear in some of the terms $\frac{\partial}{\partial S_\ell} E_k(\mathbf{x})$.

By definition, $|c_*|$ is the largest of the values $|c_\ell|$, and by setting ϵ small enough, all coefficients of $\frac{\partial}{\partial S_\ell} E_k(\mathbf{x})$ can be made negligibly small for every k. This implies that the coefficient of the monomial $\frac{\partial}{\partial S_*} p(\mathbf{x})$ can be made arbitrarily close to $c_* \cdot \frac{|S_*|!}{\sigma_d \cdot d!}$, which is nonzero since c_* is nonzero.

However, the left-hand side of equation (2.9) tells us that this coefficient should be zero - a contradiction. We conclude that $\mathbf{A}$ has full row rank, and therefore that $m_1^{\text{uniform}}(p) = m \geq \prod_{i=1}^n (r_i + 1)$. This completes the proof of part (i).

We now consider part (ii) of the theorem. It follows from Proposition 2.3.6, part (ii) that, for each i, we can Taylor-approximate $x_i^{r_i}$ using $7\lceil \log_2(r_i) \rceil$ neurons arranged in a deep network. Therefore, we can Taylor-approximate all of the $x_i^{r_i}$ using a total of $\sum_i 7\lceil \log_2(r_i) \rceil$ neurons. From [84], we know that these n terms can be multiplied using $4n$ additional neurons, giving us a total of $\sum_i (7\lceil \log_2(r_i) \rceil + 4)$. Proposition

2.2.3 implies again that $m_1^{\text{uniform}}(p) \leq m_1^{\text{Taylor}}(p)$. This completes the proof.

Proof of Theorem 2.3.2.

As above, suppose that $r_i > 0$ for $i = 1, \ldots, n$, and let X be the multiset in which x_i occurs with multiplicity r_i.

It is shown in the proof of Theorem 2.3.1 that $\prod_{i=1}^{n}(r_i+1)$ neurons are *sufficient* to Taylor-approximate $p(x)$. We now show that this number of neurons is also *necessary* for approximating $p(\mathbf{x})$. Let $m = m_1^{\text{Taylor}}(p)$ and suppose that $\sigma(x)$ has the Taylor expansion $\sum_{k=0}^{\infty} \sigma_k x^k$. Then, by grouping terms of each order, we conclude that there exist constants a_{ij} and w_j such that

$$\sigma_d \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^d = p(\mathbf{x}) \tag{2.10}$$

$$\sigma_k \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^k = 0 \quad \text{for } 0 \leq k \leq N-1. \tag{2.11}$$

For each $S \subseteq X$, let us take the derivative of equations (2.10) and (2.11) by every variable that occurs in S, where we take multiple derivatives of variables that occur multiple times. This gives

$$\frac{\sigma_d \cdot d!}{|S|!} \sum_{j=1}^{m} w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{d-|S|} = \frac{\partial}{\partial S} p(\mathbf{x}), \tag{2.12}$$

$$\frac{\sigma_k \cdot k!}{|S|!} \sum_{j=1}^{m} w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{k-|S|} = 0 \tag{2.13}$$

for $|S| \leq k \leq d-1$. Observe that there are $r = \prod_{i=1}^{n}(r_i+1)$ choices for S, since each variable x_i can be included anywhere from 0 to r_i times. Define $\mathbf{A}$ to be the $r \times m$ matrix with entries $A_{S,j} = \prod_{h \in S} a_{hj}$. We claim that $\mathbf{A}$ has full row rank. This would show that the number of columns m is at least the number of rows $r = \prod_{i=1}^{n}(r_i+1)$, proving the desired lower bound on m.

Suppose towards contradiction that the rows $A_{S_\ell,\bullet}$ admit a linear dependence:

$$\sum_{\ell=1}^{r} c_\ell A_{S_\ell,\bullet} = \mathbf{0},$$

where the coefficients c_ℓ are nonzero and the S_ℓ denote distinct subsets of X. Set $s = \max_\ell |S_\ell|$. Then, take the dot product of each side of the above equation by the vector with entries (indexed by j) equal to $w_j \left(\sum_{i=1}^{n} a_{ij}x_i\right)^{d-s}$:

$$\begin{aligned}
0 &= \sum_{\ell=1}^{r} c_\ell \sum_{j=1}^{m} w_j \prod_{h\in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij}x_i\right)^{d-s} \\
&= \sum_{\ell|(|S_\ell|=s)} c_\ell \sum_{j=1}^{m} w_j \prod_{h\in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij}x_i\right)^{d-|S_\ell|} \\
&+ \sum_{\ell|(|S_\ell|<s)} c_\ell \sum_{j=1}^{m} w_j \prod_{h\in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij}x_i\right)^{(d+|S_\ell|-s)-|S_\ell|}.
\end{aligned}$$

We can use (2.12) to simplify the first term and (2.13) (with $k = d + |S_\ell| - s$) to simplify the second term, giving us:

$$\begin{aligned}
0 &= \sum_{\ell|(|S_\ell|=s)} c_\ell \cdot \frac{|S_\ell|!}{\sigma_d \cdot d!} \cdot \frac{\partial}{\partial S_\ell} p(\mathbf{x}) + \sum_{\ell|(|S_\ell|<s)} c_\ell \cdot \frac{|S_\ell|!}{\sigma_{d+|S_\ell|-s} \cdot (d+|S_\ell|-s)!} \cdot 0 \\
&= \sum_{\ell|(|S_\ell|=s)} c_\ell \cdot \frac{|S_\ell|!}{\sigma_d \cdot d!} \cdot \frac{\partial}{\partial S_\ell} p(\mathbf{x}).
\end{aligned}$$

Since the distinct monomials $\frac{\partial}{\partial S_\ell} p(\mathbf{x})$ are linearly independent, this contradicts our assumption that the c_ℓ are nonzero. We conclude that $\mathbf{A}$ has full row rank, and therefore that $m_1^{\text{Taylor}}(p) = m \geq \prod_{i=1}^{n}(r_i + 1)$. This completes the proof of part (i).

Part (ii) of the theorem was demonstrated in the proof of Theorem 2.3.1. This completes the proof.

Proof of Theorem 2.3.3.

Our proof in Theorem 2.3.1 relied upon the fact that all nonzero partial derivatives of a monomial are linearly independent. This fact is not true for general polynomials p; however, an exactly similar argument shows that $m_1^{\text{uniform}}(p)$ is at least the number of linearly independent partial derivatives of p, taken with respect to multisets of the input variables.

Consider the monomial q of p such that $m_1^{\text{uniform}}(q)$ is maximized, and suppose that $q(\mathbf{x}) = x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}$. By Theorem 2.3.1, $m_1^{\text{uniform}}(q)$ is equal to the number $\prod_{i=1}^{n}(r_i + 1)$ of distinct monomials that can be obtained by taking partial derivatives of q. Let Q be the set of such monomials, and let D be the set of (iterated) partial derivatives corresponding to them, so that for $d \in D$, we have $d(q) \in Q$.

Consider the set of polynomials $P = \{d(p) \mid d \in D\}$. We claim that there exists a linearly independent subset of P with size at least $|D|/c$. Suppose to the contrary that P' is a maximal linearly independent subset of P with $|P'| < |D|/c$.

Since p has c monomials, every element of P has at most c monomials. Therefore, the total number of distinct monomials in elements of P' is less than $|D|$. However, there are at least $|D|$ distinct monomials contained in elements of P, since for $d \in D$, the polynomial $d(p)$ contains the monomial $d(q)$, and by definition all $d(q)$ are distinct as d varies. We conclude that there is some polynomial $p' \in P \backslash P'$ containing a monomial that does not appear in any element of P'. But then p' is linearly independent of P', a contradiction since we assumed that P' was maximal.

We conclude that some linearly independent subset of P has size at least $|D|/c$, and therefore that the space of partial derivatives of p has rank at least $|D|/c = m_1^{\text{uniform}}(q)/c$. This proves part (i) of the theorem. Part (ii) follows immediately from the definition of $m^{\text{uniform}}(p)$.

Similar logic holds for m^{Taylor}.

Proof of Theorem 2.3.4.

We will prove the desired lower bounds for $m_1^{\text{uniform}}(p)$; a very similar argument holds for $m_1^{\text{Taylor}}(p)$. As above, suppose that $r_i > 0$ for $i = 1, \ldots, n$. Let X be the multiset in which x_i occurs with multiplicity r_i.

Suppose that $N_\epsilon(\mathbf{x})$ is an ϵ-approximation to $p(\mathbf{x})$ with depth 1, and let the degree-d Taylor polynomial of $N_\epsilon(\mathbf{x})$ be $p(\mathbf{x}) + E(\mathbf{x})$. Let $E_d(\mathbf{x})$ be the degree-d homogeneous component of $E(\mathbf{x})$. Observe that the coefficients of the error polynomial $E_d(\mathbf{x})$ can be made arbitrarily small by setting ϵ sufficiently small.

Let $m = m_1^{\text{uniform}}(p)$ and suppose that $\sigma(x)$ has the Taylor expansion $\sum_{k=0}^{\infty} \sigma_k x^k$. Then, by grouping terms of each order, we conclude that there exist constants a_{ij} and w_j such that

$$\sigma_d \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^d = p(\mathbf{x}) + E_d(\mathbf{x})$$

For each $S \subseteq X$, let us take the derivative of this equation by every variable that occurs in S, where we take multiple derivatives of variables that occur multiple times. This gives

$$\frac{\sigma_d \cdot d!}{|S|!} \sum_{j=1}^{m} w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{d-|S|} = \frac{\partial}{\partial S} p(\mathbf{x}) + \frac{\partial}{\partial S} E_d(\mathbf{x}).$$

Consider this equation as $S \subseteq X$ varies over all C_s multisets of fixed size s. The left-hand side represents a linear combination of the m terms $\left(\sum_{i=1}^{n} a_{ij} x_i\right)^{d-s}$. The polynomials $\frac{\partial}{\partial S} p(\mathbf{x}) + \frac{\partial}{\partial S} E_d(\mathbf{x})$ on the right-hand side must be linearly independent as S varies, since the distinct monomials $\frac{\partial}{\partial S} p(\mathbf{x})$ are linearly independent and the coefficients of $\frac{\partial}{\partial S} E_d(\mathbf{x})$ can be made arbitrarily small.

This means that the number m of linearly combined terms on the left-hand side must be at least the number C_s of choices for S. Observe that C_s is the coefficient of the term y^s in the polynomial $g(y) = \prod_i (1 + y + \ldots + y^{r_i})$. A simple (and not very good) lower bound for C_s is $\frac{1}{d} \prod_{i=1}^{n} (r_i + 1)$, since there are $\prod_{i=1}^{n} (r_i + 1)$ distinct sub-multisets of X, and their cardinalities range from 0 to d.

Chapter 3

How to Start Training: The Effect of Initialization and Architecture

3.1 Introduction

Despite the growing number of practical uses for deep learning, training deep neural networks remains a challenge. Among the many possible obstacles to training, it is natural to distinguish two kinds: problems that prevent a given neural network from ever achieving better-than-chance performance and problems that have to do with later stages of training, such as escaping flat regions and saddle points [64, 137], reaching spurious local minima [41, 71], and overfitting [4, 161]. This paper focuses specifically on two failure modes related to the first kind of difficulty:

(FM1): The mean length scale in the final layer increases/decreases exponentially with the depth.

(FM2): The empirical variance of length scales across layers grows exponentially with the depth.

Our main contributions and conclusions are:

- **The mean and variance of activations in a neural network are both important in determining whether training begins.** If both failure modes

FM1 and FM2 are avoided, then a deeper network need not take longer to start training than a shallower network.

- **FM1 is dependent on weight initialization.** Initializing weights with the correct variance (in fully connected and convolutional networks) and correctly weighting residual modules (in residual networks) prevents the mean size of activations from becoming exponentially large or small as a function of the depth, allowing training to start for deeper architectures.

- **For fully connected and convolutional networks, FM2 is dependent on architecture.** Wider layers prevent FM2, again allowing training to start for deeper architectures. In the case of constant-width networks, the width should grow approximately linearly with the depth to avoid FM2.

- **For residual networks, FM2 is largely independent of the architecture.** Provided that residual modules are weighted to avoid FM1, FM2 can never occur. This qualitative difference between fully connected and residual networks can help to explain the empirical success of the latter, allowing deep and relatively narrow networks to be trained more readily.

FM1 for fully connected networks has been previously studied [52, 111, 134]. Training may fail to start, in this failure mode, since the difference between network outputs may exceed machine precision even for moderate d. For ReLU activations, FM1 has been observed to be overcome by initializations of He et al. [52]. We prove this fact rigorously (see Theorems 3.5.1 and 3.5.4). We find empirically that for poor initializations, training fails more frequently as networks become deeper (see Figures 3-1 and 3-4).

Aside from [149], there appears to be less literature studying FM1 for residual networks (ResNets) [53]. We prove that the key to avoiding FM1 in ResNets is to correctly rescale the contributions of individual residual modules (see Theorems 3.5.1 and 3.5.4). Without this, we find empirically that training fails for deeper ResNets (see Figure 3-2).

FM2 is more subtle and does not seem to have been widely studied (see [74] for a notable exception). We find that FM2 indeed impedes early training (see Figure 3-3). This may happen since the backpropagated SGD updates for trainable parameters at a given layer include a factor that corresponds to the activations at that layer. Thus, if lengths of activations at different layers are highly variable, then so are the SGD updates, making it difficult to choose a good learning rate for the network as a whole. Our analysis of FM2 reveals an interesting difference between fully connected and residual networks. Namely, for fully connected and convolutional networks, FM2 is a function of architecture, rather than just of initialization, and can occur even if FM1 does not. For residual networks, we prove by contrast that FM2 never occurs once FM1 is avoided (see Corollary 3.5.3 and Theorem 3.5.4).

3.2 Related Work

Closely related to this article is the work of Taki [149] on initializing ResNets. It gives heuristic computations related to the mean squared activation in a depth-L ResNet and suggests taking the scales η_ℓ of the residual modules to all be equal to $1/L$ (see §3.3.2). Also related to this work is that of He et al. [52] already mentioned above, as well as [69, 116, 118, 134]. The authors in the latter group show that information can be propagated in infinitely wide ReLU nets so long as weights are initialized independently according to an appropriately normalized distribution (see condition (ii) in Definition 3.4.1). One notable difference between this collection of papers and the present work is that we are concerned with a rigorous computation of finite width effects.

These finite size corrections were also studied by Schoenholz et al. [135], which gives exact formulas for the distribution of pre-activations in the case when the weights and biases are Gaussian. For more on the Gaussian case, we also point the reader to Giryes et al. [43]. The idea that controlling means and variances of activations at various hidden layers in a deep network can help with the start of training was previously considered in Klaumbauer et al. [74]. This work introduced the scaled

exponential linear unit (SELU) activation, which is shown to cause the mean values of neuron activations to converge to 0 and the average squared length to converge to 1. A different approach to this kind of self-normalizing behavior was suggested in Wu et al. [156]. There, the authors suggest to add a linear hidden layer (that has no learnable parameters) but directly normalizes activations to have mean 0 and variance 1. Activation lengths can also be controlled by constraining weight matrices to be orthogonal or unitary (see e.g. [3, 56, 66, 80, 133]).

Finally, we mention the previous appearance in Hanin [49] of the sum of reciprocals of layer widths, which we here show determines the variance of the sizes of the activations (see Theorem 3.5.1) in randomly initialized fully connected ReLU nets. The article [49] studied the more delicate question of the variance for gradients computed by random ReLU nets.

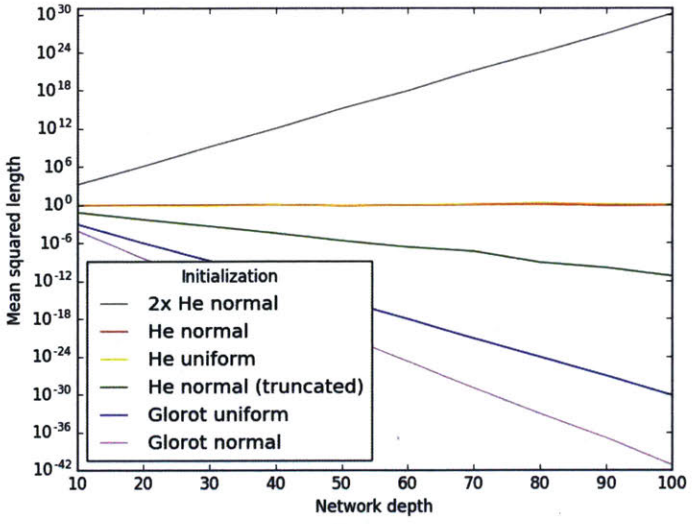
3.3 Results

In this section, we will (1) provide an intuitive motivation and explanation of our mathematical results, (2) verify empirically that our predictions hold, and (3) show by experiment the implications for training neural networks.

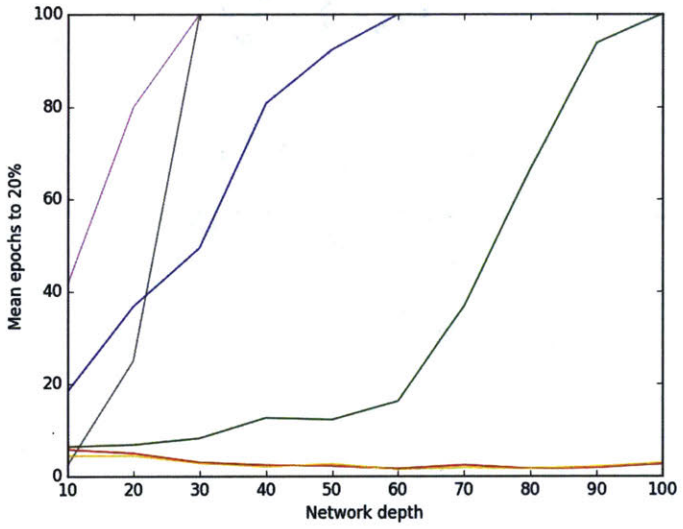
3.3.1 Avoiding FM1 for Fully Connected Networks: Variance of Weights

Consider a depth-d, fully connected ReLU net $\mathcal{N}$ with hidden layer widths n_j, $j = 0, \ldots, d$, and random weights and biases (see Definition 3.4.1 for the details of the initialization). As $\mathcal{N}$ propagates an input vector $\text{act}^{(0)} \in \mathbf{R}^{n_0}$ from one layer to the next, the lengths of the resulting vectors of activations $\text{act}^{(j)} \in \mathbf{R}^{n_j}$ change in some manner, eventually producing an output vector whose length is potentially very different from that of the input. These changes in length are summarized by

$$M_j := \frac{1}{n_j} \left\| \text{act}^{(j)} \right\|^2,$$



(a)



(b)

Figure 3-1: Comparison of the behavior of differently initialized fully connected networks as depth increases. Width is equal to depth throughout. Note that in `He normal (truncated)`, the normal distribution is truncated at two standard deviations from the mean, as implemented e.g. in Keras and PyTorch. For `2x He normal`, weights are drawn from a normal distribution with twice the variance of `He normal`. (a) Mean square length M_d (log scale), demonstrating exponential decay or explosion unless variance is set at 2/fan-in, as in `He normal` and `He uniform` initializations; (b) average number of epochs required to obtain 20% test accuracy when training on vectorized MNIST, showing that exponential decay or explosion of M_d is associated with reduced ability to begin training.

where here and throughout the squared norm of a vector is the sum of the squares of its entries. We prove in Theorem 3.5.1 that the mean of the normalized output length M_d, which controls whether failure mode FM1 occurs, is determined by the variance of the distribution used to initialize weights. We emphasize that all our results hold for *any fixed input*, which need not be random; we average only over the weights and the biases. Thus, FM1 cannot be directly solved by batch normalization [61], which renormalizes by averaging over inputs to $\mathcal{N}$, rather than averaging over initializations for $\mathcal{N}$.

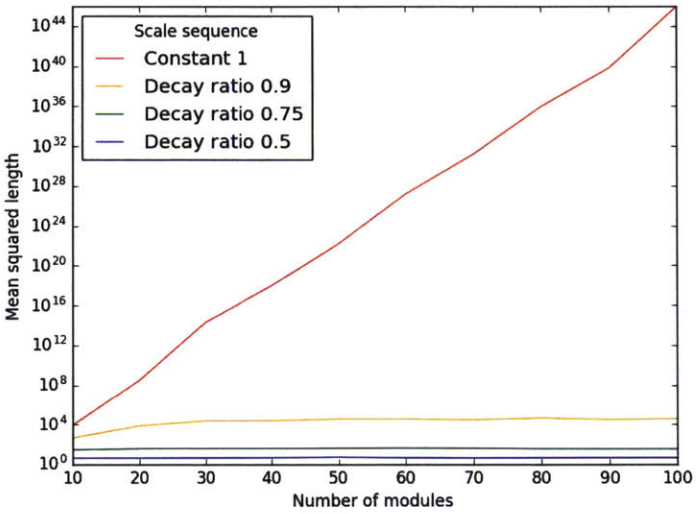
Theorem 3.3.1 (FM1 for fully connected networks (informal)). *The mean* $\mathbb{E}[M_d]$ *of the normalized output length is equal to the input length if network weights are drawn independently from a symmetric distribution with variance* $2/\textit{fan-in}$. *For higher variance, the mean* $\mathbb{E}[M_d]$ *grows exponentially in the depth* d, *while for lower variance, it decays exponentially.*

In Figure 3-1, we compare the effects of different initializations in networks with varying depth, where the width is equal to the depth (this is done to prevent FM2, see §3.3.3). Figure 3-1(a) shows that, as predicted, initializations for which the variance of weights is smaller than the critical value of 2/fan-in lead to a dramatic decrease in output length, while variance larger than this value causes the output length to explode. Figure 3-1(b) compares the ability of differently initialized networks to start training; it shows the average number of epochs required to achieve 20% test accuracy on MNIST [81]. It is clear that those initializations which preserve output length are also those which allow for fast initial training - in fact, we see that it is *faster* to train a suitably initialized depth-100 network than it is to train a depth-10 network. Datapoints in (a) represent the statistics over random unit inputs for 1,000 independently initialized networks, while (b) shows the number of epochs required to achieve 20% accuracy on vectorized MNIST, averaged over 5 training runs with independent initializations, where networks were trained using stochastic gradient descent with a fixed learning rate of 0.01 and batch size of 1024, for up to 100 epochs. Note that changing the learning rate depending on depth could be used to compensate

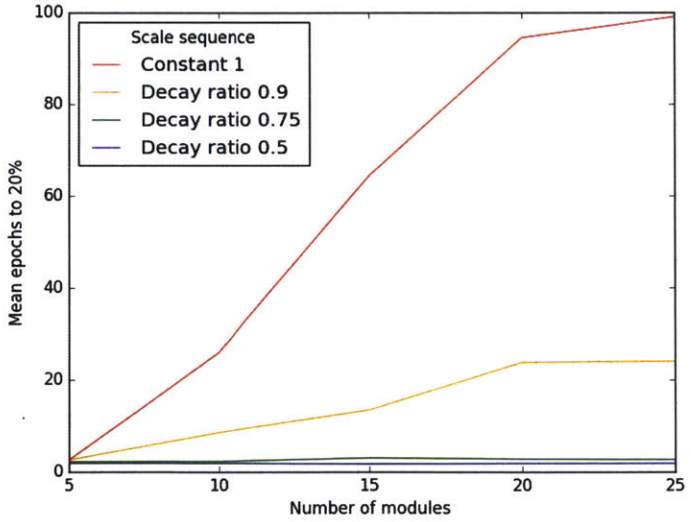
for FM1; choosing the right initialization is equivalent and much simpler.

While $\mathbb{E}[M_d]$ and its connection to the variance of weights at initialization have been previously noted (we refer the reader especially to [52] and to §3.2 for other references), the implications for choosing a good initialization appear not to have been fully recognized. Many established initializations for ReLU networks draw weights i.i.d. from a distribution for which the variance is not normalized to preserve output lengths. As we shall see, such initializations hamper training of very deep networks. For instance, as implemented in the Keras deep learning Python library [21], the only default initialization to have the critical variance 2/fan-in is `He uniform`. By contrast, `LeCun uniform` and `LeCun normal` have variance 1/fan-in, `Glorot uniform` (also known as `Xavier uniform`) and `Glorot normal` (`Xavier normal`) have variance 2/(fan-in + fan-out). Finally, the initialization `He normal` comes close to having the correct variance, but, at the time of writing, the Keras implementation truncates the normal distribution at two standard deviations from the mean (the implementation in PyTorch [108] does likewise). This leads to a decrease in the variance of the resulting weight distribution, causing a catastrophic decay in the lengths of output activations (see Figure 3-1). We note this to emphasize both the sensitivity of initialization and the popularity of initializers that can lead to FM1.

It is worth noting that the 2 in our optimal variance 2/fan-in arises from the ReLU, which zeros out symmetrically distributed input with probability 1/2, thereby effectively halving the variance at each layer. (For linear activations, the 2 would disappear.) The initializations described above may preserve output lengths for activation functions *other than ReLU*. However, ReLU is one of the most common activation functions for feed-forward networks and various initializations are commonly used blindly with ReLUs without recognizing the effect upon ease of training. An interesting systematic approach to predicting the correct multiplicative constant in the variance of weights as a function of the non-linearity is proposed in [116, 134] (e.g., the definition of χ_1 around (7) in Poole et al. [116]). For non-linearities other than ReLU, however, this constant seems difficult to compute directly.



(a)



(b)

Figure 3-2: Comparison of the behavior of differently scaled ResNets as the number of modules increases. Each residual module here is a single layer of width 5. (a) Mean length scale M_L^{res}, which grows exponentially in the sum of scales η_ℓ; (b) average number of epochs to 20% test accuracy when training on MNIST, showing that M_L^{res} is a good predictor of initial training performance.

3.3.2 Avoiding FM1 for Residual Networks: Weights of Residual Modules

To state our results about FM1 for ResNets, we must set some notation (based on the framework presented e.g. in Veit et al. [153]). For a sequence η_ℓ, $\ell = 1, 2 \ldots$ of positive real numbers and a sequence of fully connected ReLU nets $\mathcal{N}_1, \mathcal{N}_2, \ldots$, we define a *residual network* $\mathcal{N}_L^{res}$ with *residual modules* $\mathcal{N}_1, \ldots, \mathcal{N}_L$ and *scales* $\eta_1, \ldots, \eta_L$ by the recursion

$$\mathcal{N}_0^{res}(x) = x, \qquad \mathcal{N}_\ell^{res}(x) = \mathcal{N}_{\ell-1}^{res}(x) + \eta_\ell \mathcal{N}_\ell\left(\mathcal{N}_{\ell-1}^{res}(x)\right), \qquad \ell = 1, \ldots, L.$$

Explicitly,

$$\begin{aligned} \mathcal{N}_L^{res}(x) \;=\; & x \;+\; \eta_1 \mathcal{N}_1(x) \;+\; \eta_2 \mathcal{N}_2(x \;+\; \eta_1 \mathcal{N}_1(x)) \\ & +\; \eta_3 \mathcal{N}_3\left(x \;+\; \eta_1 \mathcal{N}_1(x) \;+\; \eta_2 \mathcal{N}_2\left(x \;+\; \eta_1 \mathcal{N}_1(x)\right)\right) + \cdots . \end{aligned} \tag{3.1}$$

Intuitively, the scale η_ℓ controls the size of the correction to $\mathcal{N}_{\ell-1}^{res}$ computed by the residual module $\mathcal{N}_\ell$. Since we implicitly assume that the depths and widths of the residual modules $\mathcal{N}_\ell$ are uniformly bounded (e.g., the modules may have a common architecture), failure mode FM1 comes down to determining for which sequences $\{\eta_\ell\}$ of scales there exist $c, C > 0$ so that

$$c \leq \sup_{L \geq 1} \mathbb{E}\left[M_L^{res}\right] \leq C, \tag{3.2}$$

where we write $M_L^{res} = \|\mathcal{N}_L^{res}(x)\|^2$ and x is a unit norm input to $\mathcal{N}_L^{res}$. The expectation in (3.2) is over the weights and biases in the fully connected residual modules $\mathcal{N}_\ell$, which we initialize as in Definition 3.4.1, except that we set biases to zero for simplicity (this does not affect the results below). A part of our main theoretical result, Theorem 3.5.4, on residual networks can be summarized as follows.

Theorem 3.3.2 (FM1 for ResNets (informal)). *Consider a randomly initialized ResNet with L residual modules, scales $\eta_1, \ldots, \eta_L$, and weights drawn independently from a*

symmetric distribution with variance $2/$*fan-in. The mean* $\mathbb{E}\left[M_L^{res}\right]$ *of the normalized output length grows exponentially with the sum of the scales* $\sum_{\ell=1}^{L} \eta_\ell$.

We empirically verify the predictive power of the quantity η_ℓ in the performance of ResNets. In Figure 3-2(a), we initialize ResNets with constant $\eta_\ell = 1$ as well as geometrically decaying $\eta_\ell = b^\ell$ for $b = 0.9, 0.75, 0.5$. All modules are single hidden layers with width 5. We observe that, as predicted by Theorem 3.3.1, $\eta_\ell = 1$ leads to exponentially growing length scale M_L^{res}, while $\eta_\ell = b^\ell$ leads the mean of M_L^{res} to grow until it reaches a plateau (the value of which depends on b), since $\sum \eta_\ell$ is finite. In Figure 3-2(b), we show that the mean of M_L^{res} well predicts the ease with which ResNets of different depths are trained. Note the large gap between $b = 0.9$ and 0.75, which is explained by noting that the approximation of $\eta^2 \ll \eta$ which we use in the proof holds for $\eta \ll 1$, leading to a larger constant multiple of $\sum \eta_\ell$ in the exponent for b closer to 1. Each datapoint is averaged over 100 training runs with independent initializations, with training parameters as in Figure 3-1.

3.3.3 FM2 for Fully Connected Networks: The Effect of Architecture

In the notation of §3.3.1, failure mode FM2 is characterized by a large expected value for

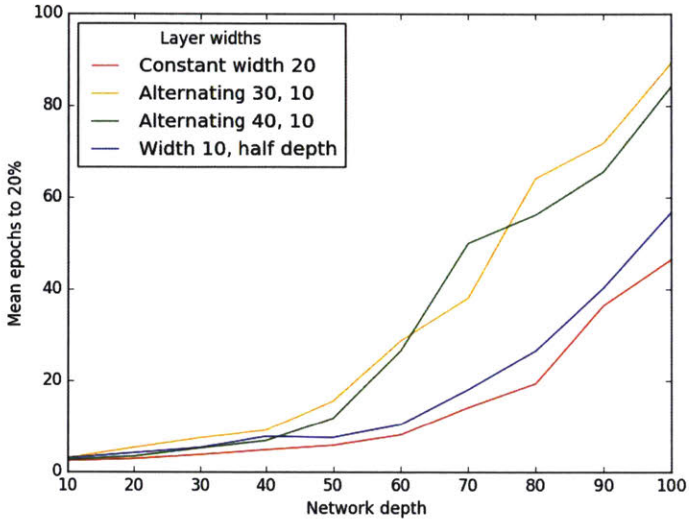
$$\widehat{\mathrm{Var}}[M] := \frac{1}{d}\sum_{j=1}^{d} M_j^2 - \left(\frac{1}{d}\sum_{j=1}^{d} M_j\right)^2,$$

the empirical variance of the normalized squared lengths of activations among all the hidden layers in $\mathcal{N}$. Our main theoretical result about FM2 for fully connected networks is the following.

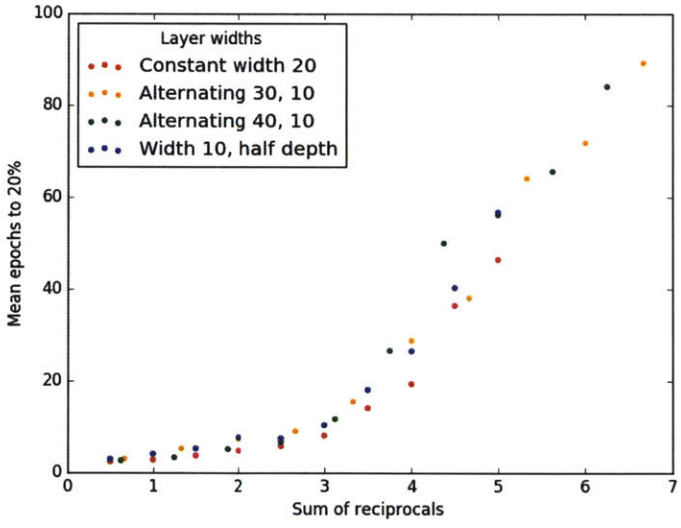
Theorem 3.3.3 (FM2 for fully connected networks (informal))**.** *The mean* $\mathbb{E}[\widehat{\mathrm{Var}}[M]]$ *of the empirical variance for the lengths of activations in a fully connected* ReLU *net is exponential in* $\sum_{j=1}^{d-1} 1/n_j$*, the sum of the reciprocals of the widths of the hidden layers.*

For a formal statement see Theorem 3.5.1. It is well known that deep but narrow networks are hard to train, and this result provides theoretical justification; since for such nets $\sum 1/n_j$ is large. More than that, this sum of reciprocals gives a definite way to quantify the effect of "deep but narrow" architectures on the volatility of the scale of activations at various layers within the network. We note that this result also implies that for a given depth and fixed budget of neurons or parameters, constant width is optimal, since by the Power Mean Inequality, $\sum_j 1/n_j$ is minimized for all n_j equal if $\sum n_j$ (number of neurons) or $\sum n_j^2$ (approximate number of parameters) is held fixed.

We experimentally verify the link between $\widehat{\text{Var}}[M]$ and speed of early training: Figure 3-3(a) compares training performance on MNIST for fully connected networks of various depths (i) with fixed width 20 (our reference network), (ii) with alternating layers of widths 30 and 10 (same number of neurons as reference net), (iii) with alternating layers of widths 40 and 10 (same number of parameters as reference net), and (iv) with fixed width 10 and half the depth (same $\sum 1/n_j$). We observe that (i) and (iv) train rapidly at the same rate, as predicted by Theorem 3.3.4, while (ii) and (iii) (which have larger $\sum 1/n_j$) train much more slowly. In all cases, training becomes harder with greater depth, since $\sum 1/n_j$ increases with depth for constant-width networks. In Figure 3-3(b), we plot the same data with $\sum 1/n_j$ on the x-axis, showing this quantity's power in predicting the effectiveness of early training, irrespective of the particular details of the network architecture in question. Each datapoint is averaged over 100 independently initialized training runs, with training parameters as in Figure 3-1. All networks are initialized with `He normal` weights to prevent FM1.



(a)



(b)

Figure 3-3: Comparison of ease with which different fully connected architectures may be trained. (a) Mean epochs required to obtain 20% test accuracy when training on MNIST, as a function of network depth; (b) same y-axis, with x-axis showing the sum of reciprocals of layer widths. Training efficiency is shown to be predicted closely by this sum of reciprocals, independent of other details of network architecture. Note that all networks are initialized with `He normal` weights to avoid FM1.

3.3.4 FM2 for Residual Networks

In the notation of §3.3.2, failure mode FM2 is equivalent to a large expected value for the empirical variance

$$\widehat{\mathrm{Var}}[M^{res}] := \frac{1}{L}\sum_{\ell=1}^{L}(M_\ell^{res})^2 - \left(\frac{1}{L}\sum_{\ell=1}^{L} M_\ell^{res}\right)^2$$

of the normalized squared lengths of activations among the residual modules in $\mathcal{N}$. Our main theoretical result about FM2 for ResNets is the following (see Theorem 3.5.1 for the precise statement).

Theorem 3.3.4 (FM2 for ResNets (informal)). *The mean* $\mathbb{E}[\widehat{\mathrm{Var}}[M^{res}]]$ *of the empirical variance for the lengths of activations in a residual* ReLU *net with* L *residual modules and scales* η_ℓ *is exponential in* $\sum_{\ell=1}^{L}\eta_\ell$. *By Theorem 3.3.2, this means that in ResNets, if failure mode FM1 does not occur, then neither does FM2 (assuming FM2 does not occur in individual residual modules).*

3.3.5 Convolutional Architectures

Our above results were stated for fully connected networks, but the logic of our proofs carries over to other architectures. In particular, similar statements hold for convolutional neural networks (ConvNets). Note that the fan-in for a convolutional layer is not given by the width of the preceding layer, but instead is equal to the number of features multiplied by the kernel size.

In Figure 3-4, we show that the output length behavior we observed in fully connected networks also holds in ConvNets. Namely, mean output length equals input length for weights drawn i.i.d. from a symmetric distribution of variance 2/fan-in, while other variances lead to exploding or vanishing output lengths as the depth increases. In our experiments, networks were purely convolutional, with no pooling or fully connected layers. By analogy to Figure 3-1, the fan-in was set to approximately the depth of the network by fixing kernel size 3×3 and setting the number of features at each layer to one tenth of the network's total depth. For each datapoint, the

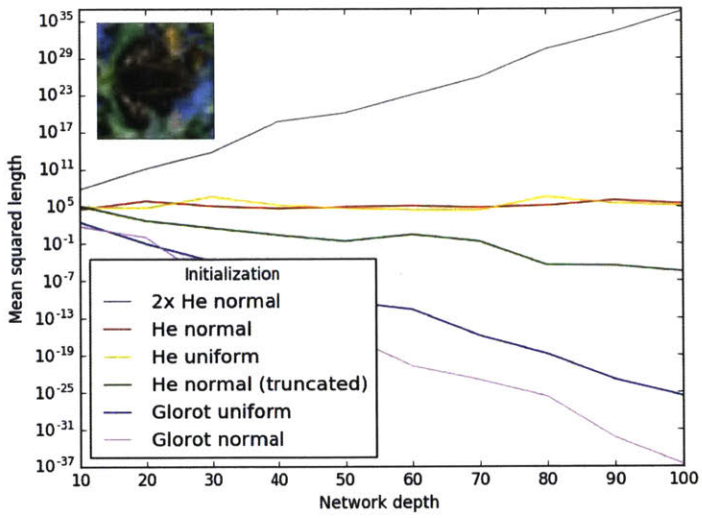


Figure 3-4: Comparison of the behavior of differently initialized ConvNets as depth increases, with the number of features at each layer proportional to the overall network depth. The mean output length over different random initializations is observed to follow the same patterns as in Figure 3-1 for fully connected networks. Weight distributions with variance 2/fan-in preserve output length, while other distributions lead to exponential growth or decay. The input image from CIFAR-10 is shown.

network was allowed to vary over 1,000 independent initializations, with input a fixed image from the dataset CIFAR-10 [78].

3.4 Notation

To state our results formally, we first give the precise definition of the networks we study; and we introduce some notation. For every $d \geq 1$ and $\mathbf{n} = (n_i)_{i=0}^{d} \in \mathbf{Z}_+^{d+1}$, we define

$$\mathfrak{N}(\mathbf{n}, d) = \left\{ \begin{array}{c} \text{fully connected feed-forward nets with ReLU activations} \\ \text{and depth } d, \text{ whose } j^{th} \text{ hidden layer has width } n_j \end{array} \right\}.$$

Note that n_0 is the dimension of the input. Given $\mathcal{N} \in \mathfrak{N}(\mathbf{n}, d)$, the function $f_{\mathcal{N}}$ it computes is determined by its weights and biases

$$\{w_{\alpha,\beta}^{(j)},\ b_{\beta}^{(j)},\ 1 \leq \alpha \leq n_j,\ \ 1 \leq \beta \leq n_{j+1},\ 0 \leq j \leq d-1\}.$$

For every input $\text{act}^{(0)} = \left(\text{act}_\alpha^{(0)}\right)_{\alpha=1}^{n_0} \in \mathbf{R}^{n_0}$ to $\mathcal{N}$, we write for all $j = 1, \ldots, d$

$$\text{preact}_\beta^{(j)} = b_\beta^{(j)} + \sum_{\alpha=1}^{n_{j-1}} \text{act}_\alpha^{(j-1)} w_{\alpha,\beta}^{(j)}, \qquad \text{act}_\beta^{(j)} = \text{ReLU}(\text{preact}_\beta^{(j)}), \quad 1 \leq \beta \leq n_j. \tag{3.3}$$

The vectors $\text{preact}^{(j)}$, $\text{act}^{(j)}$ are thus the inputs and outputs of nonlinearities in the j^{th} layer of $\mathcal{N}$.

Definition 3.4.1 (Random Nets). *Fix $d \geq 1$, positive integers $\mathbf{n} = (n_0, \ldots, n_d) \in \mathbf{Z}_+^{d+1}$, and two collections of probability measures $\mu = \left(\mu^{(1)}, \ldots, \mu^{(d)}\right)$ and $\nu = \left(\nu^{(1)}, \ldots, \nu^{(d)}\right)$ on $\mathbf{R}$ such that $\mu^{(j)}, \nu^{(j)}$ are symmetric around 0 for every $1 \leq j \leq d$, and such that the variance of $\mu^{(j)}$ is $2/(n_{j-1})$.*

A random network $\mathcal{N} \in \mathfrak{N}_{\mu,\nu}(\mathbf{n}, d)$ is obtained by requiring that the weights and biases for neurons at layer j are drawn independently from $\mu^{(j)}, \nu^{(j)}$, respectively.

3.5 Formal statements

We begin by stating our results about fully connected networks. Given a random network $\mathcal{N} \in \mathfrak{N}_{\mu,\nu}(d, \mathbf{n})$ and an input $\text{act}^{(0)}$ to $\mathcal{N}$, we write as in §3.3.1, M_j for the normalized square length of activations $\frac{1}{n_d}||\,\text{act}^{(j)}\,||^2$ at layer j. Our first theoretical result, Theorem 3.5.1, concerns both the mean and variance of M_d. To state it, we denote for any probability measure λ on $\mathbf{R}$ its moments by

$$\lambda_k := \int_{\mathbf{R}} x^k d\lambda(x).$$

Theorem 3.5.1. *For each $j \geq 0$, fix $n_j \in \mathbf{Z}_+$. For each $d \geq 1$, let $\mathcal{N} \in \mathfrak{N}_{\mu,\nu}(d, \mathbf{n})$ be a fully connected* ReLU *net with depth d, hidden layer widths $\mathbf{n} = (n_j)_{j=0}^d$ as well as random weights and biases as in Definition 3.4.1. Fix also an input $\text{act}^{(0)} \in \mathbf{R}^{n_0}$ to*

$\mathcal{N}$ *with* $||\operatorname{act}^{(0)}|| = 1$. *We have almost surely*

$$\limsup_{d\to\infty} M_d < \infty \iff \sum_{j\geq 1} \left(\nu_2^{(j)}\right)^2 < \infty. \tag{3.4}$$

Moreover, if (3.4) *holds, then exists a random variable* M_∞ *(that is almost surely finite) such that* $M_d \to M_\infty$ *as* $d \to \infty$ *pointwise almost surely. Further, suppose* $\mu_4^{(j)} < \infty$ *for all* $j \geq 1$ *and that* $\sum_{j=1}^{\infty} \left(\nu_2^{(j)}\right)^4 < \infty$. *Then there exist* $C, N > 0$ *so that if* $n_j \geq N$ *for all* j, *then*

$$\exp\left[\frac{1}{2}\sum_{j=1}^{d}\frac{1}{n_j}\right] \leq \mathbb{E}\left[M_d^2\right] \leq C\exp\left[C\sum_{j=1}^{d}\frac{1}{n_j}\right]. \tag{3.5}$$

In particular, $\operatorname{Var}[M_d]$ *is exponential in* $\sum_{j=1}^{d} 1/n_j$ *and if* $\sum_j 1/n_j < \infty$, *then the convergence of* M_d *to* M_∞ *is in* L^2 *and* $\operatorname{Var}[M_\infty] < \infty$.

The proof of Theorem 3.5.1 is given in §3.6. Although we state our results only for fully connected feed-forward ReLU nets, the proof techniques carry over essentially verbatim to any feed-forward network in which only weights in the same hidden layer are tied. In particular, our results apply to convolutional networks in which the kernel sizes are uniformly bounded. In this case, the constants in Theorem 3.5.1 depend on the bound for the kernel dimensions, and n_j denotes the fan-in for neurons in the $(j+1)^{st}$ hidden layer (i.e. the number of channels in layer j multiplied by the size of the appropriate kernel). We also point out the following corollary, which follows immediately from the proof of Theorem 3.5.1.

Corollary 3.5.2 (FM1 for Fully Connected Networks)**.** *With notation as in Theorem 3.5.1, suppose that for all* $j = 1, \ldots, d$, *the weights in layer* j *of* $\mathcal{N}_d$ *have variance* $\kappa \cdot 2/n_j$ *for some* $\kappa > 0$. *Then the average squared size* M_d *of activations at layer* d *will grow or decay exponentially unless* $\kappa = 1$*:*

$$\mathbb{E}\left[\frac{1}{n_d}\left\|\operatorname{act}^{(d)}\right\|^2\right] = \frac{\kappa^d}{n_0}\left\|\operatorname{act}^{(0)}\right\|^2 + \sum_{j=1}^{d}\kappa^{d-j}\nu_2^{(d)}.$$

Our final result about fully connected networks is a corollary of Theorem 3.5.1, which explains precisely when failure mode FM2 occurs (see §3.3.3). It is proved in §3.7.

Corollary 3.5.3 (FM2 for Fully Connected Networks). *Take the same notation as in Theorem 3.5.1. There exist $c, C > 0$ so that*

$$\frac{c}{d}\sum_{j=1}^{d}\frac{j-1}{d}\exp\left(c\sum_{k=j}^{d-1}\frac{1}{n_k}\right) \quad\leq\quad \mathbb{E}\left[\widehat{\mathrm{Var}}[M]\right] \quad\leq\quad \frac{C}{d}\sum_{j=1}^{d}\frac{j-1}{d}\exp\left(C\sum_{k=j}^{d-1}\frac{1}{n_k}\right). \tag{3.6}$$

In particular, suppose the hidden layer widths are all equal, $n_j = n$. Then, $\mathbb{E}[\widehat{\mathrm{Var}}[M]]$ is exponential in $\beta = \sum_j 1/n_j = (d-1)/n$ in the sense that there exist $c, C > 0$ so that

$$c\exp(c\beta) \quad\leq\quad \mathbb{E}\left[\widehat{\mathrm{Var}}[M]\right] \quad\leq\quad C\exp(C\beta). \tag{3.7}$$

Finally, our main result about residual networks is the following (proven in §3.8):

Theorem 3.5.4. *Take the notation from §3.3.2 and §3.3.4. The mean of squared activations in $\mathcal{N}_L^{res}$ are uniformly bounded in the number of modules L if and only if the scales η_ℓ form a convergent series:*

$$0 < \sup_{L\geq 1,\ \|x\|=1} M_L^{res} < \infty \qquad\Longleftrightarrow\qquad \sum_{\ell=1}^{\infty}\eta_\ell < \infty. \tag{3.8}$$

Moreover, for any sequence of scales η_ℓ for which $\sup_\ell \eta_\ell < 1$ and for every $K, L \geq 1$, we have

$$\mathbb{E}\left[(M_L^{res})^K\right] = \exp\left(O\left(\sum_{\ell=1}^{L}\eta_\ell\right)\right),$$

where the implied constant depends on K but not on L. Hence, once the condition in part (3.8) holds, both the moments $\mathbb{E}\left[(M_L^{res})^K\right]$ and the mean of the empirical variance of $\{M_\ell^{res},\ \ell = 1, \ldots, L\}$ are uniformly bounded in L.

3.6 Proof of Theorem 3.5.1

Let us first verify that M_d is a submartingale for the filtration $\{\mathcal{F}_d\}_{d\geq 1}$ with $\mathcal{F}_d$ being the sigma algebra generated by all weights and biases up to and including layer d (for background on sigma algebras and martingales we refer the reader to Chapters 2 and 37 in [15]). Since $\text{act}^{(0)}$ is a fixed non-random vector, it is clear that M_d is measurable with respect to $\mathcal{F}_d$. We have

$$\begin{aligned}\mathbb{E}\left[M_d \mid \mathcal{F}_{d-1}\right] &= \frac{1}{n_d}\mathbb{E}\left[\left\|\text{act}^{(d)}\right\|^2 \mid \text{act}^{(d-1)}\right] \\ &= \frac{1}{n_d}\sum_{\beta=1}^{n_d}\mathbb{E}\left[\left(\text{preact}_\beta^{(d)}\right)^2 \mathbf{1}_{\{\text{preact}_\beta^{(d)}>0\}} \mid \text{act}^{(d-1)}\right], \end{aligned} \tag{3.9}$$

where we can replace the sigma algebra $\mathcal{F}_{d-1}$ by the sigma algebra generated by $\text{act}^{(d-1)}$ since the computation done by a feed-forward neural net is a Markov chain with respect to activations at consecutive layers (for background see Chapter 8 in [15]). Next, recall that by assumption the weights and biases are symmetric in law around 0. Note that for each β, changing the signs of all the weights $w_{\alpha,\beta}^{(d)}$ and biases $b_\beta^{(d)}$ causes $\text{preact}_\beta^{(d)}$ to change sign. Hence, we find

$$\mathbb{E}\left[\left(\text{preact}_\beta^{(d)}\right)^2 \mathbf{1}_{\{\text{preact}_\beta^{(d)}>0\}} \mid \text{act}^{(d-1)}\right] = \mathbb{E}\left[\left(\text{preact}_\beta^{(d)}\right)^2 \mathbf{1}_{\{\text{preact}_\beta^{(d)}<0\}} \mid \text{act}^{(d-1)}\right].$$

Note that

$$\text{preact}_\beta^{(d)}\left(\mathbf{1}_{\{\text{preact}_\beta^{(d)}>0\}} + \mathbf{1}_{\{\text{preact}_\beta^{(d)}<0\}}\right) = \text{preact}_\beta^{(d)}.$$

Symmetrizing the expression in (3.9), we obtain

$$\begin{aligned}\mathbb{E}\left[M_d \mid \mathcal{F}_{d-1}\right] &= \frac{1}{2n_d}\sum_{\beta=1}^{n_d}\mathbb{E}\left[\left(\text{preact}_\beta^{(d)}\right)^2 \mid \text{act}^{(d-1)}\right] \\ &= \frac{1}{2n_d}\sum_{\beta=1}^{n_d}\mathbb{E}\left[\left(b_\beta^{(d)} + \sum_{\alpha=1}^{n_{d-1}}\text{act}_\alpha^{(d-1)}\, w_{\alpha,\beta}^{(d)}\right)^2 \mid \text{act}^{(d-1)}\right] \\ &= \frac{1}{2}\left(\nu_2^{(d)}\right)^2 + \frac{1}{n_{d-1}}\left\|\text{act}^{(d-1)}\right\|^2 \geq M_{d-1}, \end{aligned} \tag{3.10}$$

where in the second equality we used that the weights $w_{\alpha,\beta}^{(d)}$ and biases $b_\beta^{(d)}$ are independent of $\mathcal{F}_{d-1}$ with mean 0 and in the last equality that $\mathrm{Var}[w_{\alpha,\beta}^{(d)}] = 2/n_{j-1}$. The above computation also yields that for each $d \geq 1$,

$$\mathbb{E}\left[M_d\right] = \mathbb{E}\left[\frac{1}{n_d}\left\|\mathrm{act}^{(d)}\right\|^2\right] = \frac{1}{n_0}\left\|\mathrm{act}^{(0)}\right\|^2 + \frac{1}{2}\sum_{j=1}^{d}\left(\nu_2^{(j)}\right)^2. \tag{3.11}$$

It also shows that $\widehat{M}_d = M_d - \sum_{j=1}^{d}\frac{1}{2}\nu_2^{(d)}$ is a martingale. Taking the limit $d \to \infty$ in (3.11) proves (3.4). Next, assuming condition (3.4), we find that

$$\sup_{d\geq 1}\mathbb{E}\left[\max\{M_d, 0\}\right] \leq \frac{1}{n_0}\left\|\mathrm{act}^{(0)}\right\|^2 + \frac{1}{2}\sum_{j=1}^{\infty}\left(\nu_2^{(j)}\right)^2,$$

which is finite. Hence, we may apply Doob's pointwise martingale convergence theorem (see Chapter 35 in [15]) to conclude that the limit

$$M_\infty = \lim_{d\to\infty} M_d$$

is exists and is finite almost surely. To show (3.5) we will need the following result.

Lemma 3.6.1. *There exists $C > 0$ so that for every $d \geq 1$*

$$\frac{M_{d-1}^2}{n_d} \leq \mathrm{Var}[M_d \mid \mathcal{F}_{d-1}] \leq \frac{C(1 + M_{d-1}^2 + M_{d-1})}{n_d}$$

Proof. Note that

$$M_d = \frac{1}{n_d}\sum_\beta\left(\mathrm{act}_\beta^{(d)}\right)^2,$$

and, conditioned on $\mathrm{act}^{(d-1)}$, the random variables $\{\mathrm{act}_\beta^{(d)}\}_\beta$ are i.i.d. Hence,

$$\mathrm{Var}[M_d \mid \mathcal{F}_{d-1}] = \frac{1}{n_d}\mathrm{Var}\left[\left(\mathrm{act}_1^{(d)}\right)^2 \mid \mathcal{F}_{d-1}\right]. \tag{3.12}$$

Since $\mathbb{E}\left[M_{d-1}\right]$ is bounded above by a uniform constant, we will be done once we show

for some $C > 0$

$$M_{d-1}^2 \leq \mathrm{Var}\left[\left(\mathrm{act}_1^{(d)}\right)^2 \mid \mathcal{F}_{d-1}\right] \leq C(1 + M_{d-1}^2 + M_{d-1}). \tag{3.13}$$

We apply the same symmetrization trick as in the derivation of (3.10) to obtain

$$\begin{aligned}\mathbb{E}\left[\left(\mathrm{act}_1^{(d)}\right)^4 \mid \mathcal{F}_{d-1}\right] &= \frac{1}{2}\mathbb{E}\left[\left(\mathrm{preact}_1^{(d)}\right)^4 \mid \mathrm{act}^{(d-1)}\right] \\ &= \frac{1}{2}\mathbb{E}\left[\left(\sum_{\alpha=1}^{n_{d-1}} \mathrm{act}_\alpha^{(d-1)} w_{\alpha,1}^{(d)} + b_1^{(d)}\right)^4 \mid \mathrm{act}^{(d-1)}\right],\end{aligned}$$

which after using that the odd moments of $w_{\alpha,1}^{(d)}$ and $b_1^{(d)}$ vanish becomes

$$\frac{1}{2}\mathbb{E}\left[\left(\sum_{\alpha=1}^{n_{d-1}} \mathrm{act}_\alpha^{(d-1)} w_{\alpha,1}^{(d)}\right)^4 \mid \mathrm{act}^{(d-1)}\right] + \frac{6\nu_2^{(d)}}{n_{d-1}}\left\|\mathrm{act}^{(d-1)}\right\|^2 + \frac{1}{2}\nu_4^{(d)}.$$

To evaluate the first term, note that

$$\mathbb{E}\left[\left(\sum_{\alpha=1}^{n_{d-1}} \mathrm{act}_\alpha^{(d-1)} w_{\alpha,1}^{(d)}\right)^4 \mid \mathrm{act}^{(d-1)}\right] = \sum_{\substack{\alpha_i=1 \\ 1\leq i\leq 4}}^{n_{d-1}} \prod_{i=1}^4 \mathrm{act}_{\alpha_i}^{(d-1)} \mathbb{E}\left[\prod_{i=1}^4 w_{\alpha_i,1}^{(d)}\right].$$

Since

$$\mathbb{E}\left[\prod_{i=1}^4 w_{\alpha_i,1}^{(d)}\right] = \frac{4}{n_{d-1}^2}\left[\mathbf{1}_{\substack{\{\alpha_1=\alpha_2\} \\ \{\alpha_3=\alpha_4\}}} + \mathbf{1}_{\substack{\{\alpha_1=\alpha_3\} \\ \{\alpha_2=\alpha_4\}}} + \mathbf{1}_{\substack{\{\alpha_1=\alpha_4\} \\ \{\alpha_2=\alpha_3\}}} + (\widetilde{\mu}_4^{(d)} - 3)\mathbf{1}_{\{\alpha_1=\alpha_2=\alpha_3=\alpha_4\}}\right],$$

we conclude that

$$\mathbb{E}\left[\left(\sum_{\alpha=1}^{n_{d-1}} \mathrm{act}_\alpha^{(d-1)} w_{\alpha,1}^{(d)}\right)^4 \mid \mathrm{act}^{(d-1)}\right] = \frac{4}{n_{d-1}^2}\left(3\left\|\mathrm{act}^{(d-1)}\right\|^4 + \left(\widetilde{\mu}_4^{(d)} - 3\right)\left\|\mathrm{act}^{(d-1)}\right\|_4^4\right).$$

Putting together the preceding computations and using that

$$\mathbb{E}\left[\left(\mathrm{act}_\beta^{(d)}\right)^2 \mid \mathrm{act}^{(d-1)}\right] = \frac{1}{n_{d-1}}\left\|\mathrm{act}^{(d-1)}\right\|^2 + \frac{1}{2}\nu_2^{(d)},$$

we find that is

$$\operatorname{Var}\left[\left(\operatorname{act}_1^{(d)}\right)^2 \mid \mathcal{F}_{d-1}\right] = \frac{5}{n_{d-1}^2}\left\|\operatorname{act}^{(d-1)}\right\|^4 + \frac{2(\widetilde{\mu}_4^{(d)}-3)}{n_{d-1}^2}\left\|\operatorname{act}^{(d-1)}\right\|_4^4 + \frac{5\nu_2^{(d)}}{n_{d-1}}\left\|\operatorname{act}^{(d-1)}\right\|^2 + \frac{1}{2}\nu_4^{(d)} - \frac{1}{4}\left(\nu_2^{(d)}\right)^2.$$

Recall that the excess kurtosis $\widetilde{\mu}_4^{(d)} - 3$ of $\mu^{(d)}$ is bounded below by -2 for any probability measure (see Chapter 4 in [138]) and observe that $||\operatorname{act}^{(d-1)}||_4^4 \leq ||\operatorname{act}^{(d-1)}||^4$. Therefore, using that $\frac{1}{2}\nu_4^{(d)} - \frac{1}{4}(\nu_2^{(d)})^2 \geq 0$, we obtain

$$M_{d-1}^2 \leq \operatorname{Var}\left[\left(\operatorname{act}_1^{(d)}\right)^2 \mid \mathcal{F}_{d-1}\right] \leq C(1 + M_{d-1}^2 + M_{d-1})$$

for some $C > 0$. This is precisely (3.13) and completes the proof of the Lemma. □

To conclude the proof of Theorem 3.5.1, we write

$$\mathbb{E}\left[M_d^2 \mid \mathcal{F}_{d-1}\right] = \operatorname{Var}[M_d \mid \mathcal{F}_{d-1}] + \left(M_{d-1} + \frac{1}{2}\nu_2^{(d)}\right)^2$$

and combine Lemma 3.6.1 with the expression (3.11) to obtain

$$\mathbb{E}\left[M_d^2 \mid \mathcal{F}_{d-1}\right] \leq \frac{C}{n_d}\left(1 + M_{d-1}^2 + M_d\right) + M_{d-1}^2 + M_{d-1}\nu_2^{(d)} + \frac{1}{4}\nu_4^{(d)}$$

and

$$\mathbb{E}\left[M_d^2 \mid \mathcal{F}_{d-1}\right] \geq M_{d-1}^2\left(1 + \frac{1}{n_d}\right).$$

Taking expectations of both sides in the inequalities above yields that for some $C > 0$

$$\mathbb{E}\left[M_{d-1}^2\right]\left(1 + \frac{1}{n_d}\right) \leq \mathbb{E}\left[M_d^2\right] \leq \left(Ca_d + \mathbb{E}\left[M_{d-1}^2\right]\right)\left(1 + \frac{C}{n_d}\right),$$

where $a_d = \frac{\nu_4^{(d)}}{4} + \nu_2^{(d)}$. Iterating the lower bound in this inequality yields the lower bound in (3.5). Similarly, using that $1 + C/n_d > 1$, we iterate the upper bound to

obtain

$$\begin{aligned}\mathbb{E}\left[M_d^2\right] &\leq \left(Ca_d + \mathbb{E}\left[M_{d-1}^2\right]\right)\left(1+\frac{C}{n_d}\right)\\ &\leq \left(C(a_d+a_{d-1}) + \mathbb{E}\left[M_{d-2}^2\right]\right)\left(1+\frac{C}{n_d}\right)\left(1+\frac{C}{n_{d-1}}\right)\\ &\cdots \leq \left(C\sum_{j=1}^{d} a_j + M_0\right)\exp\left(C\sum_{j=1}^{d}\frac{1}{n_j}\right).\end{aligned}$$

Using that $\sum_j a_j < \infty$, this gives the upper bound in (3.5) and completes the proof of Theorem 3.5.1.

3.7 Proof of Corollary 3.5.3

Fix a fully connected ReLU net $\mathcal{N}$ with depth d and hidden layer widths $n_0, \ldots, n_d$. We fix an input $\mathrm{act}^{(0)}$ to $\mathcal{N}$ and study the empirical variance $\widehat{\mathrm{Var}}[M]$ of the squared sizes of activations M_j, $j = 1, \ldots, d$. Since the biases in $\mathcal{N}$ are 0, the squared activations M_j are a martingale (see (3.10)) and we find

$$\mathbb{E}\left[M_j M_{j'}\right] = \mathbb{E}\left[M_{\min\{j,j'\}}^2\right].$$

Thus, using that by (3.5) for some $c > 0$

$$\mathbb{E}\left[M_j^2\right] \geq c\exp\left(c\sum_{k=j}^{d-1}\frac{1}{n_k}\right),$$

we find

$$\begin{aligned}\mathbb{E}\left[\widehat{\mathrm{Var}}_d\right] &= \frac{1}{d}\sum_{j=1}^{d}\mathbb{E}\left[M_j^2\right] - \frac{1}{d^2}\sum_{j,j'=1}^{d}\mathbb{E}\left[M_j M_{j'}\right]\\ &= \frac{1}{d}\sum_{j=1}^{d}\mathbb{E}\left[M_j^2\right] - \frac{1}{d^2}\sum_{j=1}^{d}(d-j+1)\mathbb{E}\left[M_j^2\right]\\ &\geq \frac{1}{d}\sum_{j=1}^{d}\frac{j-1}{d}c\exp\left(c\sum_{k=j}^{d-1}\frac{1}{n_k}\right).\end{aligned}$$

To see that this sum is exponential in $\sum 1/n_j$ as in (3.7), let us consider the special case of equal widths $n_j = n$. Then, writing

$$\beta = \sum_{j=1}^{d-1} \frac{1}{n_j},$$

we have

$$\sum_{j=1}^{d} \frac{j-1}{d} c \exp\left(c \sum_{k=j}^{d-1} \frac{1}{n_k}\right) \approx \int_0^1 x e^{\beta x} dx \geq \frac{1}{4} e^{\beta/2}.$$

This proves the lower bounds in (3.6) and (3.7). The upper bounds are similar.

3.8 Proof of Theorem 3.5.4

To understand the sizes of activations produced by $\mathcal{N}_L^{res}$, we need the following Lemma.

Lemma 3.8.1. *Let $\mathcal{N}$ be a feed-forward, fully connected* ReLU *net with depth d and hidden layer widths $n_0, \ldots, n_d$ having random weights as in Definition 3.4.1 and biases set to 0. Then for each $\eta \in (0,1)$, we have*

$$\|x + \eta \mathcal{N}(x)\|^2 = \|x\|^2 (1 + O(\eta)).$$

Proof of Lemma. We have:

$$\begin{aligned} \mathbb{E}\left[\|x + \eta \mathcal{N}(x)\|\right]^2 &= \mathbb{E}\left[\|x\|^2 + 2\eta \langle x, \mathcal{N}(x)\rangle + \eta^2 \|\mathcal{N}(x)\|^2\right] \\ &= \|x\|^2 \left(1 + \eta^2 + 2\eta \mathbb{E}\left[\langle \widehat{x}, \mathcal{N}(\widehat{x})\rangle\right]\right), \end{aligned} \tag{3.14}$$

where $\widehat{x} = \frac{x}{\|x\|}$, and we have used the fact that $\|\mathcal{N}(x)\|^2 = \|x\|^2$ (see (3.10)) as well as the positive homogeneity of ReLU nets with zero biases:

$$\mathcal{N}(\lambda x) = \lambda \mathcal{N}(x), \qquad \lambda > 0.$$

Write

$$\mathbb{E}\left[\langle x, \mathcal{N}(x)\rangle\right] = \sum_{\beta=1}^{n} x_\beta \mathbb{E}\left[\mathcal{N}_\beta(x)\right].$$

Let us also write $x = x^{(0)}$ for the input to $\mathcal{N}$, similarly set $x^{(j)}$ for the activations at layer j. We denote by $W_\beta^{(j)}$ the β^{th} row of the weights $W^{(j)}$ at layer j in $\mathcal{N}$. We have:

$$\begin{aligned}
\mathbb{E}\left[\mathcal{N}_\beta(x) \mid x^{(d-1)}\right] &= \mathbb{E}\left[x_\beta^{(d)}\right] = \mathbb{E}\left[W_\beta^{(d)} x^{(d-1)} \mathbf{1}_{\{W_\beta^{(d-1)} x\}>0} \mid x^{(d-1)}\right] \\
&= \mathbb{E}\left[\left|W_\beta^{(d)} x^{(d-1)}\right| \mathbf{1}_{\{W_\beta^{(d)} x^{(d-1)}>0\}} \mid x^{(d-1)}\right] \\
&= \frac{1}{2}\mathbb{E}\left[\left|W_\beta^{(d)} x^{(d-1)}\right| \mid x^{(d-1)}\right] \\
&\leq \frac{1}{2}\left(\mathbb{E}\left[\left|W_\beta^{(d)} x^{(d-1)}\right|^2 \mid x^{(d-1)}\right]\right)^{1/2} \\
&= \frac{1}{\sqrt{2n}}\left\|x^{(d-1)}\right\|.
\end{aligned}$$

Therefore, using that $\left\|x^{(j)}\right\|$ is a supermartingale (since its square is a martingale by (3.10)):

$$\mathbb{E}\left[\mathcal{N}_\beta(x)\right] \leq \frac{1}{\sqrt{2n}}\mathbb{E}\left[\left\|x^{(d-1)}\right\|\right] \leq \frac{1}{\sqrt{2n}}\left\|x^{(0)}\right\| = \frac{1}{\sqrt{2n}}\|x\|.$$

Hence, we obtain:

$$\mathbb{E}\left[\langle \widehat{x}, \mathcal{N}(\widehat{x})\rangle\right] = O\left(\frac{\|\widehat{x}\|_1}{\sqrt{n}}\right) = O(1)$$

since by Jensen's inequality,

$$\sum_{j=1}^{n} |x_i| \leq \sqrt{n}\sum_{j=1}^{n} x_i^2, \qquad x_i \in \mathbb{R}.$$

Combining this with (3.14) completes the proof. □

The Lemma implies part (i) of the Theorem as follows:

$$\begin{aligned}\mathbb{E}\left[M_L^{res}\right] &= \mathbb{E}\left[\left\|\mathcal{N}_{L-1}^{res}(x) + \eta_L \mathcal{N}_L\left(\mathcal{N}_{L-1}^{res}(x)\right)\right\|^2\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[\left\|\mathcal{N}_{L-1}^{res}(x) + \eta_L \mathcal{N}_L\left(\mathcal{N}_{L-1}^{res}(x)\right)\right\|^2 \;\middle|\; \mathcal{N}_{L-1}^{res}(x)\right]\right] \\ &= \left(1 + O\left(\eta_L\right)\right)\mathbb{E}\left[M_{L-1}^{res}\right],\end{aligned}$$

where we used the fact that $\eta_\ell^2 = O(\eta_\ell)$ since $\eta_\ell \in (0,1)$. Iterating this inequality yields

$$\mathbb{E}\left[M_L^{res}\right] = \prod_{\ell=1}^{L}\left(1 + O\left(\eta_\ell\right)\right) = \exp\left(\sum_{\ell=1}^{L}\log\left(1 + O\left(\eta_\ell\right)\right)\right) = \exp\left(O\left(\sum_{\ell=1}^{L}\eta_\ell\right)\right),$$

Derivation of the estimates (ii) follows exactly the same procedure and hence is omitted. Finally, using these estimates, we find that the mean empirical variance of $\{M_\ell^{res}\}$ is exponential in $\sum_\ell \eta_\ell$:

$$\begin{aligned}\mathbb{E}\left[\frac{1}{L}\sum_{\ell=1}^{L}(M_\ell^{res})^2 - \left(\frac{1}{L}\sum_{\ell=1}^{L}M_\ell^{res}\right)^2\right] &\leq \frac{1}{L}\sum_{\ell=1}^{L}\mathbb{E}\left[(M_\ell^{res})^2\right] \\ &= \frac{1}{L}\sum_{\ell=1}^{L}\exp\left(O\left(\sum_{j=1}^{\ell}\eta_j\right)\right) \\ &= \exp\left(O\left(\sum_{j=1}^{L}\eta_j\right)\right).\end{aligned}$$

3.9 Conclusion

In this article, we give a rigorous analysis of the layerwise length scales in fully connected, convolutional, and residual ReLU networks at initialization. We find that a careful choice of initial weights is needed for well-behaved mean length scales. For fully connected and convolutional networks, this entails a critical variance for i.i.d. weights, while for residual nets this entails appropriately rescaling the residual modules. For fully connected nets, we prove that to control not merely the mean but also the

variance of layerwise length scales requires choosing a sufficiently wide architecture, while for residual nets nothing further is required. We also demonstrate empirically that both the mean and variance of length scales are strong predictors of early training dynamics. In the future, we plan to extend our analysis to other (e.g. sigmoidal) activations, recurrent networks, weight initializations beyond i.i.d. (e.g. orthogonal weights), and the joint distributions of activations over several inputs.

Chapter 4

Deep Learning is Robust to Massive Label Noise

4.1 Introduction

Deep neural networks are typically trained using supervised learning on large, carefully annotated datasets. However, the need for such datasets restricts the space of problems that can be addressed. This has led to a proliferation of deep learning results on the same tasks using the same well-known datasets. However, carefully annotated data is difficult to obtain, especially for classification tasks with large numbers of classes (requiring extensive annotation) or with fine-grained classes (requiring skilled annotation). Thus, annotation can be expensive and, for tasks requiring expert knowledge, may simply be unattainable at scale.

To address this limitation, other training paradigms have been investigated to alleviate the need for expensive annotations, such as unsupervised learning [80], self-supervised learning [114, 157] and learning from noisy annotations [68, 103, 154]. Very large datasets (e.g., [76, 152]) can often be obtained, for example from web sources, with partial or unreliable annotation. This can allow neural networks to be trained on a much wider variety of tasks or classes and with less manual effort. The good performance obtained from these large, noisy datasets indicates that deep learning approaches can tolerate modest amounts of noise in the training set.

In this work, we study the behavior of deep neural networks under extremely low label reliability, only slightly above chance. The insights from our study can help guide future settings in which arbitrarily large amounts of data are easily obtainable, but in which labels come without any guarantee of validity and may merely be biased towards the correct distribution.

The key takeaways from this chapter may be summarized as follows:

- **Deep neural networks are able to generalize after training on massively noisy data, instead of merely memorizing noise.** We demonstrate that standard deep neural networks still perform well even on training sets in which label accuracy is as low as 1 percent above chance. On MNIST, for example, performance still exceeds 90 percent even with this level of label noise (see Figure 4-1). This behavior holds, to varying extents, across datasets as well as patterns of label noise, including when noisy labels are biased towards confused classes.

- **A sufficiently large training set can accommodate a wide range of noise levels.** We find that the minimum dataset size required for effective training increases with the noise level (see Figure 4-9). A large enough training set can accommodate a wide range of noise levels. Increasing the dataset size further, however, does not appreciably increase accuracy (see Figure 4-8).

- **High levels of label noise decrease the effective batch size**, as noisy labels roughly cancel out and only a small learning signal remains. As such, dataset noise can be partly compensated for by larger batch sizes and by scaling the learning rate with the effective batch size.

4.2 Related Work

Learning from noisy data. Several studies have investigated the impact of noisy datasets on machine classifiers. Approaches to learn from noisy data can generally be categorized into two groups: In the first group, approaches aim to learn directly

from noisy labels and focus on noise-robust algorithms, e.g., [10, 46, 68, 76, 89, 97, 151]. The second group comprises mostly label-cleansing methods that aim to remove or correct mislabeled data, e.g., Brodley and Friedl [17]. Methods in this group frequently face the challenge of disambiguating between mislabeled and hard training examples. To address this challenge, they often use semi-supervised approaches by combining noisy data with a small set of clean labels [163]. Some approaches model the label noise as conditionally independent from the input image [102, 144] and some propose image-conditional noise models [152, 158]. Our work differs from these approaches in that we do not aim to clean the training dataset or propose new noise-robust training algorithms. Instead, we study the behavior of standard neural network training procedures in settings with massive label noise. We show that even without explicit cleaning or noise-robust algorithms, neural networks can learn from data that has been diluted by an arbitrary amount of label noise.

Analyzing the robustness of neural networks. Several investigative studies aim to improve our understanding of convolutional neural networks. One particular stream of research in this space seeks to investigate neural networks by analyzing their robustness. For example, Veit, Wilber, and Belongie [153] show that network architectures with residual connections have a high redundancy in terms of parameters and are robust to the deletion of multiple complete layers during test time. Further, Szegedy et al. [147] investigate the robustness of neural networks to adversarial examples. They show that even for fully trained networks, small changes in the input can lead to large changes in the output and thus misclassification. In contrast, we are focusing on non-adversarial noise during training time. Within this stream of research, closest to our work are studies that focus on the impact of noisy training datasets on classification performance (e.g., [144, 151, 161]). In these studies an increase in noise is assumed to decrease not only the *proportion* of correct examples, but also their *absolute number*. In contrast to these studies, we separate the effects and show in §4.4 that a decrease in the number of correct examples is more destructive to learning than an increase in the number of noisy labels.

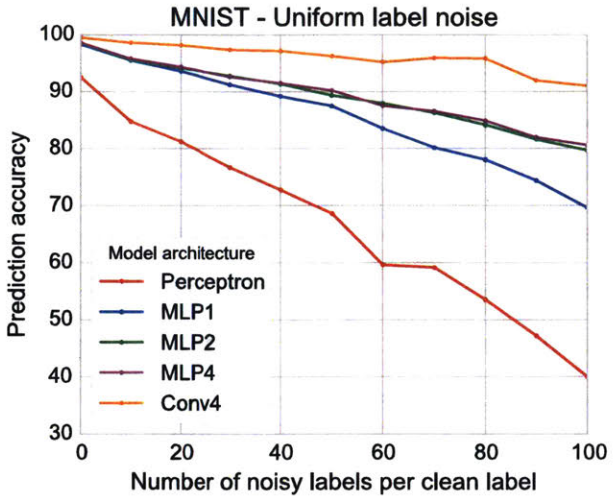


Figure 4-1: Performance on MNIST as different amounts of noisy labels are added to a fixed training set of clean labels. We compare a perceptron, MLPs with 1, 2, and 4 hidden layers, and a 4-layer ConvNet. Even with 100 noisy labels for every clean label the ConvNet still attains a performance of 91%.

4.3 Learning with massive label noise

In this work, we are concerned with scenarios of abundant data of very poor label quality, i.e., the regime in which falsely labeled training examples vastly outnumber correctly labeled examples. In particular, our experiments involve observing the performance of deep neural networks on multi-class classification tasks as label noise is increased.

To formalize the problem, we denote the number of original training examples by n. To model the amount of noise, we dilute the dataset by adding α noisy examples to the training set for each original training example. Thus, the total number of noisy labels in the training set is αn. Note that by varying the noise level α, we do not change the available number of original examples. Thus, even in the presence of high noise, there is still appreciable data to learn from, if we are able to pick it out. This is in contrast to previous work (e.g., [144, 151, 161]), in which an increase in noise also implies a decrease in the absolute number of correct examples. In the following experiments we investigate three different types of label noise: uniform label-swapping, structured label-swapping, and out-of-vocabulary examples. Thus,

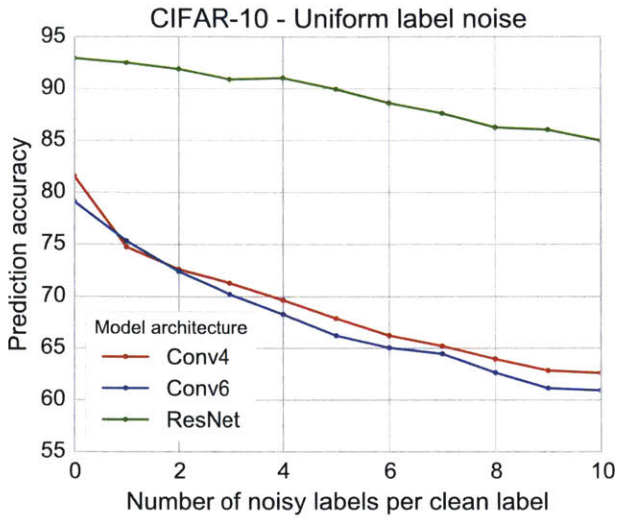


Figure 4-2: Performance on CIFAR-10 as different amounts of noisy labels are added to a fixed training set of clean labels. We tested ConvNets with 4 and 6 layers, and a ResNet with 101 layers. Even with 10 noisy labels for every clean label the ResNet still attains a performance of 85%.

the noisy label is allowed to be dependent on the correct class but not on the image itself.

A key assumption in this chapter is that unreliable labels are better modeled by an unknown stochastic process rather than by the output of an adversary. This is a natural assumption for data that is pulled from the environment, in which antagonism is not to be expected in the noisy annotation process. Deep neural networks have been shown to be exceedingly brittle to adversarial noise patterns [147]. In this work, we demonstrate that even massive amounts of non-adversarial noise present far less of an impediment to learning.

4.3.1 Experiment 1: Training with uniform label noise

As a first experiment, we will show that common training procedures for neural networks are resilient even to settings where correct labels are outnumbered by labels sampled uniformly at random at a ratio of 100 to 1. For this experiment we focus on the task of image classification and work with three commonly used datasets, MNIST [81], CIFAR-10 [78] and ImageNet [35].

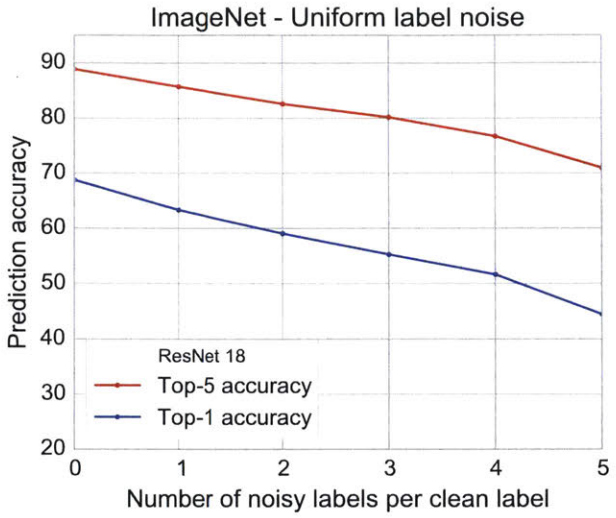


Figure 4-3: Performance on ImageNet as different amounts of noisy labels are added to a fixed training set of clean labels. Even with 5 noisy labels for every clean label, the 18-layer ResNet still attains a performance of 70%.

In Figures 4-1 and 4-2 we show the classification performance with varying levels of label noise. For MNIST, we vary the ratio α of randomly labeled examples to cleanly labeled examples from 0 (no noise) to 100 (only 11 out of 101 labels are correct, as compared with 10.1 for pure chance). For the more challenging dataset CIFAR-10, we vary α from 0 to 10. For the most challenging dataset ImageNet, we let α range from 0 to 5. We compare various architectures of neural networks: multilayer perceptrons with different numbers of hidden layers, convolutional networks (ConvNets) with different numbers of convolutional layers, and residual networks (ResNets) with different numbers of layers [53]. We evaluate performance after training on a test dataset that is free from noisy labels. Full details of our experimental setup are provided in §4.3.4.

Our results show that, remarkably, it is possible to attain over 90 percent accuracy on MNIST, even when there are 100 randomly labeled images for every cleanly labeled example, to attain over 85 percent accuracy on CIFAR-10 with 10 random labels for every clean label, and to attain over 70 percent top-5 accuracy on ImageNet with 5 random labels for every clean label. Thus, in this high-noise regime, deep networks are able not merely to perform above chance, but to attain accuracies that would be respectable even without noise.

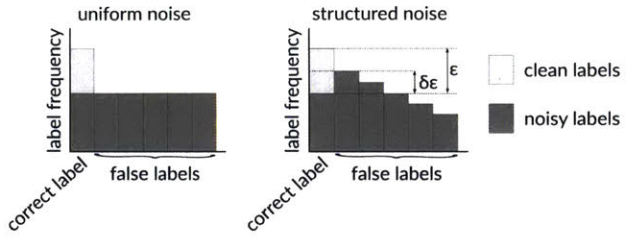


Figure 4-4: Illustration of uniform and structured noise models. In the case of structured noise, the order of false labels is important; we tested decreasing order of confusion, increasing order of confusion, and random order. The parameter δ parameterizes the degree of structure in the noise. It defines how much more likely the second most likely class is over chance.

Further, we observe from Figures 4-1 and 4-2 that larger neural network architectures tend also to be more robust to label noise. On MNIST, the performance of a perceptron decays rapidly with increasing noise (though it still attains 40 percent accuracy, well above chance, at $\alpha = 100$). The performance of a multilayer perceptron drops off more slowly, and the ConvNet is even more robust. Likewise, for CIFAR-10, the accuracy of the residual network drops more slowly than that of the smaller ConvNets. This observation provides further support for the effectiveness of ConvNets and ResNets in particular for applications where noise tolerance may be important.

4.3.2 Experiment 2: Training with structured label noise

We have seen that neural networks are extremely robust to uniform label noise. However, label noise in datasets gathered from a natural environment is unlikely to follow a perfectly uniform distribution. In this experiment, we investigate the effects of various forms of structured noise on the performance of neural networks. Figure 4-4 illustrates the procedure used to model noise structure.

In the uniform noise setting, as illustrated on the left side of Figure 4-4, correct labels are more likely than any individual false label. However, overall false labels vastly outnumber correct labels. We denote the likelihood over chance for a label to be correct as ϵ. Note that $\epsilon = 1/(1 + \alpha)$, where α is the ratio of noisy labels to certainly correct labels. To induce structure in the noise, we bias noisy labels to

certain classes. We introduce the parameter δ to parameterize the degree of structure in the noise. It defines how much more likely the second most likely class is over chance. With $\delta = 0$ the noise is uniform, whereas for $\delta = 1$ the second most likely class is equally likely as the correct class. The likelihood for the remaining classes is scaled linearly, as illustrated in Figure 4-4 on the right. We investigate three different setups for structured noise: labels biased towards easily confused classes, towards hardly confused classes and towards random classes.

Figure 4-5 shows the results on MNIST for the three different types of structured noise, as δ varies from 0 to 1. In this experiment, we train 4-layer ConvNets on a dataset that is diluted with 20 noisy labels for each clean label. We vary the order of false labels so that, besides the correct class, labels are assigned most frequently to (1) those most often confused with the correct class, (2) those least often confused with it, and (3) in a random order. We determine commonly confused labels by training the network repeatedly on a small subset of MNIST and observing the errors it makes on a test set.

The results show that deep neural nets are robust even to structured noise, as long as the correct label remains the most likely by at least a small margin. Generally, we do not observe large differences between the different models of noise structure, only that bias towards random classes seems to hurt the performance a little more than bias towards confused classes. This result might help explain why we often observe quite good results from real world noisy datasets, where label noise is more likely to be biased towards related and confusing classes.

4.3.3 Experiment 3: Source of noisy labels

In the preceding experiments, we diluted the training sets with noisy examples drawn from the same dataset; i.e., falsely labeled examples were images from within other categories of the dataset. In natural scenarios, however, noisy examples likely also include categories not included in the dataset that have erroneously been assigned labels within the dataset.

Thus, we now consider two alternative sources for noisy training examples. First,

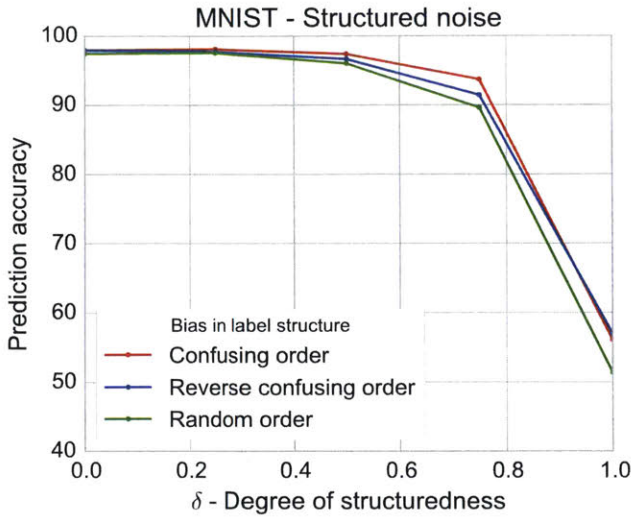


Figure 4-5: Performance on MNIST with fixed $\alpha = 20$ noisy labels per clean label. Noise is drawn from three types of structured distribution: (1) "confusing order" (highest probability for the most confusing label), (2) "reverse confusing order", and (3) random order. We interpolate between uniform noise, $\delta = 0$, and noise so highly skewed that the most common false label is as likely as the correct label, $\delta = 1$. Except for $\delta \approx 1$, performance is similar to uniform noise.

we dilute the training set with examples that are drawn from a similar but different dataset. In particular, we use CIFAR-10 as our training dataset and dilute it with examples from CIFAR-100, assigning each image a category from CIFAR-10 at random. Second, we also consider a dilution of the training set with "examples" that are simply white noise; in this case, we match the mean and variance of pixels within CIFAR-10 and again assign labels uniformly at random.

Figure 4-6 shows the results obtained by a six-layer ConvNet on the different noise sources for varying levels of noise. We observe that both alternative sources of noise lead to better performance than the noise originating from the same dataset. For noisy examples drawn from CIFAR-100, performance drops only about half as much as when noise originates from CIFAR-10 itself. This trend is consistent across noise levels. For white noise, performance does not drop regardless of noise level; this is in line with prior work that has shown that neural networks are able to fit random input [161]. This indicates the scenarios considered in Experiments 1 and 2 represent in some sense a worst case.

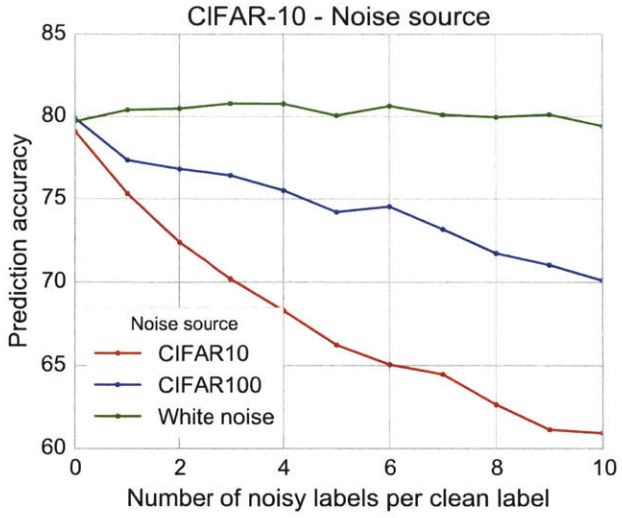


Figure 4-6: Performance on CIFAR-10 for varying amounts of noisy labels from different sources. Noisy training examples are drawn from (1) CIFAR-10 itself, but mislabeled uniformly at random, (2) CIFAR-100, with uniformly random labels, and (3) white noise with mean and variance chosen to match those of CIFAR-10. Noise drawn from CIFAR-100 resulted in only half the drop in performance observed with noise from CIFAR-10 itself, while white noise examples did not appreciable affect performance.

In natural scenarios, we may expect massively noisy datasets to fall somewhere in between the cases exemplified by CIFAR-10 and CIFAR-100. That is, some examples will be relevant but mislabeled. However, it is likely that many examples will not be from any classes under consideration and therefore will influence training less negatively. In fact, it is possible that such examples might increase accuracy, if the erroneous labels reflect underlying similarity between the examples in question.

4.3.4 Experimental setup

All models are trained with AdaDelta [160] as optimizer and a batch size of 128. For each level of label noise we train separate models with different learning rates in $\{0.01, 0.05, 0.1, 0.5\}$ and pick the learning rate that results in the best performance. Generally, we observe that the higher the label noise, the lower the optimal learning rate. We investigate this trend in detail in §4.5.

In Experiments 1 and 2, noisy labels are drawn from the same dataset as the

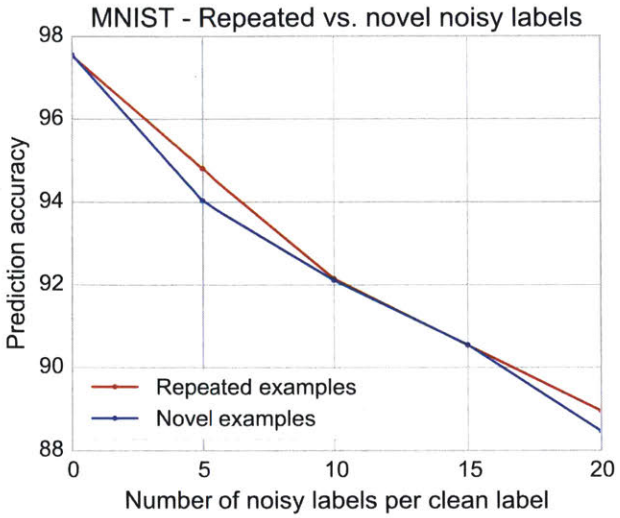


Figure 4-7: Comparison of the effect of reusing images vs. using novel images as noisy examples. Essentially no difference is observed between the two types of noisy examples, supporting the use of repeated examples in our experiments.

labels guaranteed to be correct. This involves drawing the same example many times from the dataset, giving it the correct label once, and in every other instance picking a random label according to the noise distribution in question. We show in Figure 4-7 that performance would have been comparable had we been able to draw noisy labels from an extended dataset, instead of repeating images. Specifically, we train a convolutional network on a subset of MNIST, with 2,500 certainly correct labels and with noisy labels drawn either with repetition from this set of 2,500 or without repetition from the remaining examples in the MNIST dataset. The results are essentially identical between repeated and unique examples, supporting our setup in the preceding experiments.

4.4 The importance of larger datasets

Underlying the ability of deep networks to learn from noisy data is the size of the data in question. It is well-established, see e.g., [35, 146], that traditional deep learning relies upon large datasets. We will now see how this is particularly true of noisy datasets.

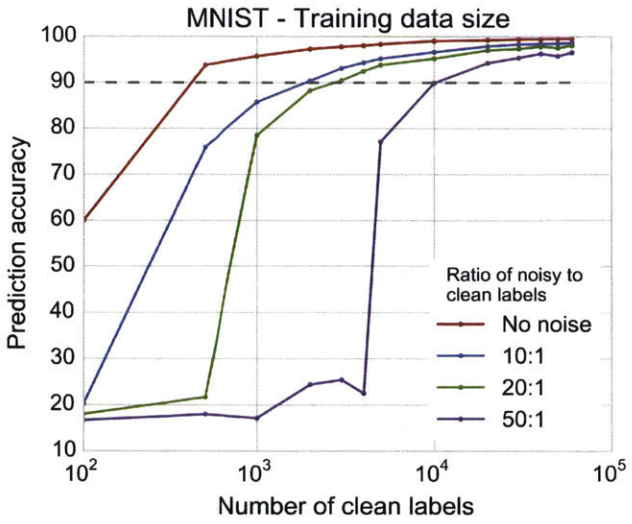


Figure 4-8: Performance on MNIST at various noise levels, as a function of the number of clean labels. There seems to be a critical amount of clean training data required to successfully train the networks. This threshold increases as the noise level rises. For example, at $\alpha = 10$, 2,000 clean labels are needed to attain 90% performance, while at $\alpha = 50$, 10,000 clean labels are needed.

In Figure 4-8, we compare the performance of a ConvNet on MNIST as the size of the training set varies. In particular, we show how the performance of the ConvNet varies with the number of *cleanly labeled training examples.* We compare the performance of the same ConvNet trained on MNIST diluted with different amounts of noisy labels sampled uniformly. For example, for the blue curve of $\alpha = 10$ and 1,000 clean labels, the network is trained on 11,000 examples: 1,000 cleanly labeled examples and 10,000 with random labels. In Figure 4-10, we show that a similar pattern occurs when a ResNet with 18 layers is trained on ImageNet with noisy labels.

Generally, we observe that independent of the noise level the networks benefit from more data and that, given sufficient data, the networks reach similar results. Further, the results indicate that there seems to be a critical amount of clean training data that is required to successfully train the networks. This critical amount of clean data depends on the noise level; in particular, it increases as the noise level rises. Note that as the amount of clean data increases, the size of the overall noisy training set increases still faster. In Figure 4-9, we re-plot the data from Figure 4-8 to show how

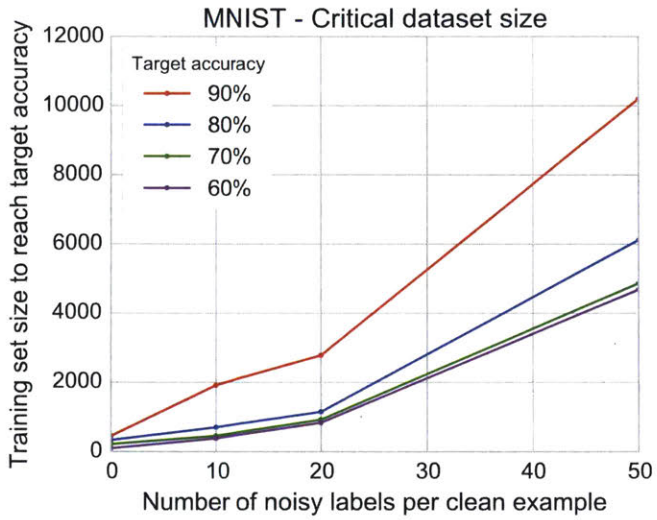


Figure 4-9: Relationship between the amount of noise in the dataset and the critical number of clean training examples needed to achieve high test accuracy. Different curves reflect different target accuracies. We observe that the required amount of clean data increases slightly more than linearly in the ratio of noisy to clean data.

the amount of clean data required to attain a threshold accuracy varies as a function of noise. It appears that the requisite amount of clean data rises slightly faster than linear in the ratio of noisy to clean examples. Since performance rapidly levels off past the critical threshold, the main requirement for the clean training set is to be of sufficient size.

4.5 Analysis

The success of training with substantial label noise comes as a surprise. Stochastic gradient descent and its variations consist of walks through the space of networks in locally optimal directions. Adding noise to this process means that in many cases the step taken is in an erroneous direction. It is well known (see e.g. [137, 64, 71, 41]) that the space of networks has abundant hard-to-escape saddle points, flat regions, and local optima; therefore, it might be supposed that too many erroneous steps could lead to a part of the space from which further optimization becomes impossible. Our result that networks generalize even when trained with almost entirely incorrect labels

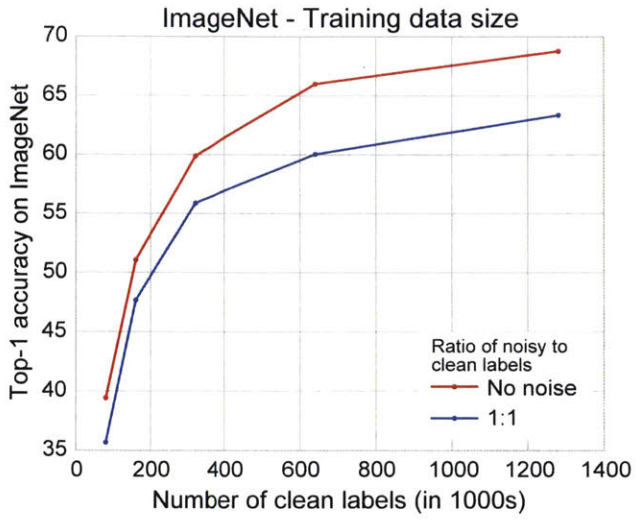


Figure 4-10: Performance on ImageNet at various noise levels, as a function of the number of clean labels, when training a ResNet with 18 layers. As in Figure 4-8, we observe that noise increases the critical number of clean training examples needed to achieve high accuracy. Once past the critical threshold for dataset size, accuracy plateaus and additional training examples are not of significant marginal utility.

suggests that trajectories in network space are more flexible than might be expected.

In this section, we analyze one aspect of the standard learning procedure which reduces the stochasticity of the gradient updates: averaging gradients over a batch. In the preceding sections, our results were obtained by training neural networks with fixed batch size and running a parameter search to pick the optimal learning rate from five possible values. We now look in more detail into how different batch sizes and learning rates affect learning on noisy datasets.

In Figure 4-11, we compare the performance of a simple 2-layer ConvNet on MNIST with increasing noise, as batch size varies from 32 to 256. (Note that all other experiments in this chapter are performed with a fixed batch size of 128.) We observe that increasing the batch size provides greater robustness to noisy labels. One possible explanation for this behavior is that within a batch, gradient updates from randomly sampled noisy labels roughly cancel out, while gradients from correct examples that are marginally more frequent sum together and contribute to learning. By this logic, large batch sizes are more robust to noise since the mean gradient over a larger batch is closer to the gradient for correct labels.

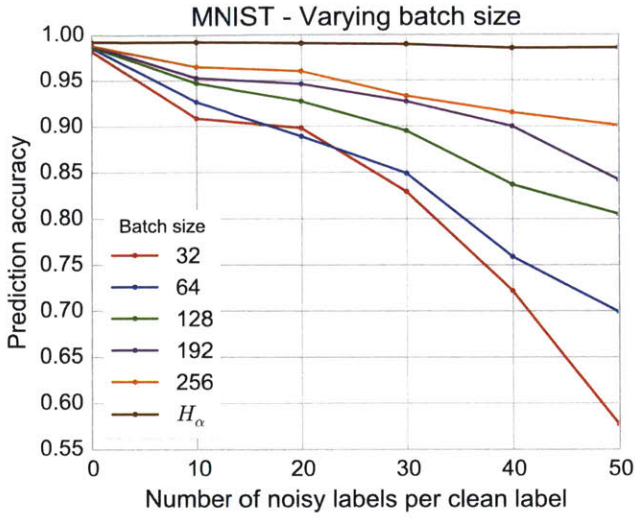


Figure 4-11: Performance on MNIST using a 2-layer ConvNet for varying batch size as a function of noise level. Higher batch size gives better performance, reflecting our analysis that increasing amounts of noise reduce the *effective* batch size. We approximate the limit of infinite batch size by training without noisy labels, using instead the *noisy loss function* H_α (see (4.3)).

To investigate this explanation further, we also consider the theoretical case of *infinite* batch size, in which gradients are averaged over the entire space of possible inputs at each training step. While this is often impossible to perform in practice, we can simulate such behavior by an auxiliary loss function.

In classification tasks, we are given an input $\mathbf{x}$ and aim to predict the class $f(\mathbf{x}) \in \{1, 2, \ldots, m\}$. The value $f(\mathbf{x})$ is encoded within a neural network by the 1-hot vector $\mathbf{y}(\mathbf{x})$ such that

$$y_k(\mathbf{x}) = \begin{cases} 1 & \text{if } k = f(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

for $1 \leq k \leq m$. Then, the standard cross-entropy loss over a batch X is given by:

$$H(X) = -\langle \log \hat{y}_{f(\mathbf{x})} \rangle_X, \tag{4.2}$$

where $\hat{\mathbf{y}}$ is the predicted vector and $\langle \cdot \rangle_X$ denotes the expected value over the batch X. We assume $\hat{\mathbf{y}}$ is normalized (e.g. by the softmax function) so that the entries sum

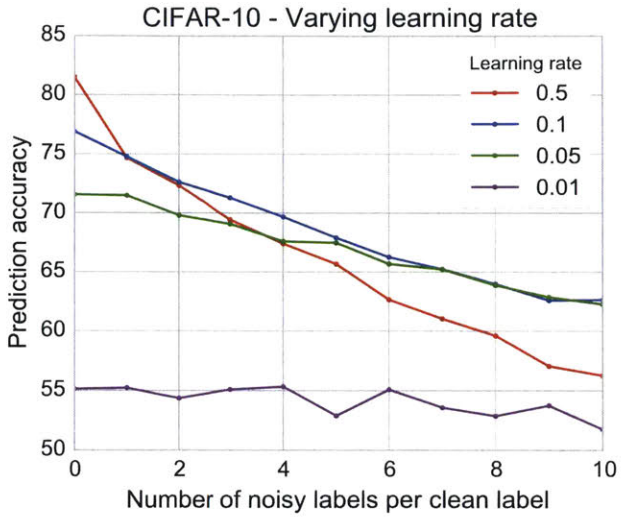


Figure 4-12: Performance on CIFAR-10 using a 4-layer ConvNet for varying learning rate as a function of noise level. Lower learning rates are generally optimal as the noise level increases.

to 1.

For a training set with noisy labels, we may consider the label $f(\mathbf{x})$ given in the training set to be merely an approximation to the true label $f_0(x)$. Consider the case of n training examples, and αn noisy labels that are sampled uniformly at random from the set $\{1, 2, \ldots, m\}$. Then, $f(\mathbf{x}) = f_0(\mathbf{x})$ with probability $\frac{1}{1+\alpha}$, and otherwise it is $1, 2, \ldots, m$, each with probability $\frac{\alpha}{m(1+\alpha)}$. As batch size increases, the expected value over the batch X is approximated more closely by these probabilities. In the limit of infinite batch size, equation (4.2) takes the form of a *noisy loss function* H_α:

$$\begin{aligned} H_\alpha(X) &:= -\frac{1}{1+\alpha}\langle \log \hat{y}_{f_0(\mathbf{x})} \rangle_X - \frac{\alpha}{m(1+\alpha)} \sum_{k=1}^{m} \langle \log \hat{y}_k \rangle_X \\ &\propto -\langle \log \hat{y}_{f_0(\mathbf{x})} \rangle_X - \alpha \left\langle \log \prod_{k=1}^{m} \hat{y}_k^{1/m} \right\rangle_X \end{aligned} \tag{4.3}$$

We can therefore compare training using the cross-entropy loss with αn noisy labels to training using the noisy loss function H_α without noisy labels. The term on the right-hand side of (4.3) represents the noise contribution, and is clearly minimized where $\hat{y}_k$ are all equal. As α increases, this contribution is weighted more heavily

against $-\langle \log \hat{y}_{f_0(\mathbf{x})} \rangle_X$, which is minimized at $\hat{\mathbf{y}}(\mathbf{x}) = \mathbf{y}(\mathbf{x})$.

We show in Figure 4-11 the results of training our 2-layer ConvNet on MNIST with the noisy loss function H_α, simulating αn noisy labels with infinite batch size. We can observe that the network's accuracy does not decrease as α increases. This can be explained by the observation that for large batch-sizes increasing α is merely decreasing the magnitude of the true gradient, rather than altering its direction.

Our observations indicate that increasing noise in the training set *reduces the effective batch size*, as noisy signals roughly cancel out and only small learning signal remains. Thus, increasing the batch size is a simple practical means to mitigate the effect of noisy training labels.

It has become common practice in training deep neural networks to scale the learning rate with the batch size. In particular, it has been shown that the smaller the batch size, the lower the optimal learning rate [77]. As noisy labels reduce the effective batch size, we would expect that lower learning rates perform better than large learning rates as noise increases. Figure 4-12 shows the performance of a 4-layer ConvNet trained with different learning rates on CIFAR-10 for varying label noise. As expected, we observe that the optimal learning rate decreases as noise increases. For example, the optimal learning rate for the clean dataset is 1, while, with the introduction of noise, this learning rate becomes unstable.

4.6 Conclusion

In this chapter, we studied the behavior of deep neural networks on training sets with very noisy labels. In a series of experiments, we have demonstrated that learning is robust to an essentially arbitrary amount of label noise, provided that the number of clean labels is sufficiently large. We have further shown that the required number of clean labels increases slightly above linear with the ratio of noisy to clean labels. Finally, we have observed that noisy labels reduce the effective batch size, an effect that can be mitigated by larger batch sizes and downscaling the learning rate.

It is worthy of note that although deep networks appear robust to even high

degrees of label noise, clean labels still always perform better than noisy labels, given the same quantity of training data. Further, one still requires expert-vetted test sets for evaluation. Lastly, it is important to reiterate that our studies focus on non-adversarial noise.

Our work suggests numerous directions for future investigation. For example, we are interested in how label-cleaning and semi-supervised methods affect the performance of networks in a high-noise regime. Are such approaches able to lower the threshold for training set size? Finally, it remains to translate the results we present into an actionable trade-off between data annotation and acquisition costs, which can be utilized in real world training pipelines for deep networks on massive noisy data.

Chapter 5

Neuron Dendrograms Uncover Asymmetrical Motifs

5.1 Motivation

The branching morphologies of neurons have fascinated neuroscientists since the imagery of Ramón y Cajal. Nevertheless, we know little about their statistical structure. The branching structure of a neuron morphology can be captured in the *dendrogram* (see Figure 5-1), which represents the neuron as a graph theoretical tree. While certain factors surrounding branching are environmental or arise from interactions with other neurons, it is believed that intrinsic biological factors, which may differ across neuron types, affect the branching of neurons during development. By statistically analyzing the branching structures of different types of neurons, we hope to shed light on the biological hidden variables that may affect branching processes.

5.2 Definitions

We define four variations on the basic Galton-Watson branching processes to model neuron dendrograms: symmetric, asymmetric, depth-dependent symmetric, and depth-dependent asymmetric. The symmetric model is the basic Galton-Watson process, in which a dendrogram grows by each node branching with probability p, and otherwise

terminating. In the *asymmetric* models, each node has a hidden type (A or B), and the probability of its branching depends on its type: either p_A or p_B. When a node branches, one child is given the type A and the other has type B. If p_A is high and p_B is low, then nodes with label A correspond to "main branches" that persist as the tree grows, while those with label B correspond to "side branches" that are less likely to persist. In the *depth-dependent* models, the probability of branching is also dependent on the distance from the root node, representing the soma of the neuron.

5.3 Comparing motifs

To further motivate our more careful statistical analysis below, we calculated the frequency of certain simple motifs as subgraphs within the dendograms of five types of neurons (or parts thereof). Figure 5-2 shows that asymmetric motifs are overall more frequent, while different types of neurons have markedly different frequency patterns. For example, granular cells exhibit more symmetrical motifs than the average over all types of neurons. All data used was obtained from `neuromorpho.org`.

5.4 Comparing statistical models

To compare our four models, we used tenfold cross-validation on datasets representing several classes of neurons (see Figure 5-3). For a given neuron class and branching model, we calculated the difference between the log likelihood of obtaining dendrograms from that neuron class using the given model and the log likelihood using the depth-dependent asymmetric model. The mean and variance were calculated by dividing the data into training and testing in ten different ways. The cross-validation procedure, in which models are evaluated only on data which they are not trained on, allows us to conclude that the depth-dependent asymmetric model gives the best fit for every neuron class considered. However, it is worth noting that axons (not shown), which typically branch relatively little, are better represented by more symmetric models.

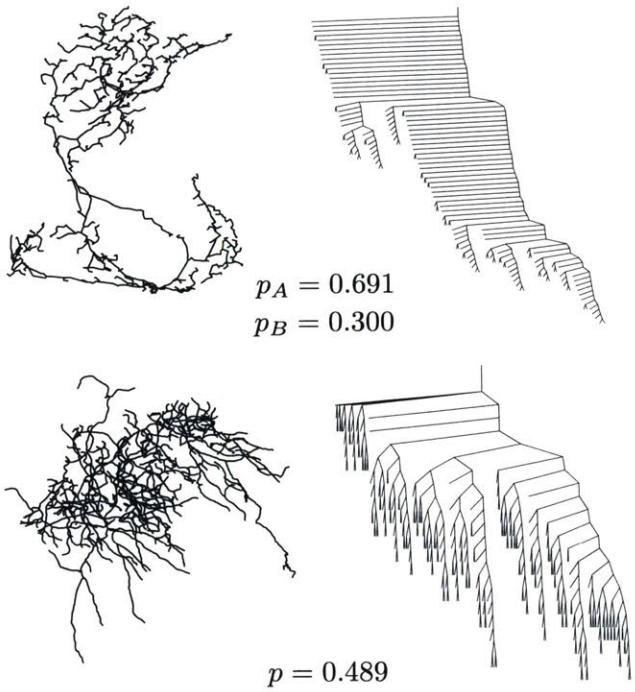


Figure 5-1: Neuron morphologies, shown geometrically (left) and through dendrograms (right). Above, a neuron with fitted asymmetric parameters p_A and p_B shown (note that the two probabilities are very different, indicating asymmetry). Below, a more symmetric neuron, with parameter p shown.

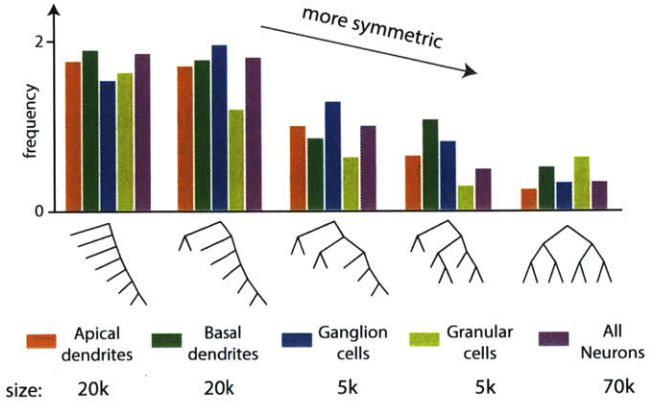


Figure 5-2: Frequency of graphical motifs within dendrograms of various classes of neurons (or parts thereof). Apical and basal dendrites are drawn from pyramidal neurons. The motifs are sorted such that their probabilities under more symmetric models grow from left to right. The size of each database is given under the neuron type.

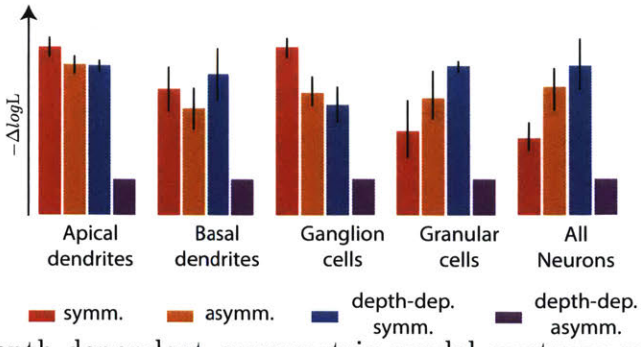


Figure 5-3: The depth-dependent asymmetric model captures real neuron dendrograms most effectively, by comparing other models to it using cross-validation.

5.5 Fitting statistical models

To fit each model, we calculate the likelihood of obtaining the observed dendrograms using that model, and fit branching probabilities so as to maximize the likelihood. For the symmetric model, the likelihood for a single dendrogram can be expressed by multiplying p for each branching and $1-p$ for each termination, giving the formula $p^{n^+}(1-p)^{n^-}$, where n^+ is the number of branching nodes (those with 2 children) and n^- is the number of terminating nodes. In the asymmetric model, the likelihood is given by:

$$[p_A p_B]^{n^b} \cdot [p_A(1-p_B) + p_B(1-p_A)]^{n^c} \cdot [(1-p_A)(1-p_B)]^{n^d}$$

where n^b, n^c, n^d are, respectively, the numbers of branching nodes for which both children, one child, and neither child is branching. Similar computations are possible for depth-dependent models, though they are slightly more complicated. The fitted parameters for depth-dependent models over the entire databases of various neuron types are shown in Figure 5-4.

5.6 Conclusions

1. Branching motifs within neuron morphologies differ markedly across neuron types. Such motifs could be helpful in distinguishing neuron types automatically, as well as in suggesting biological differences between such types.

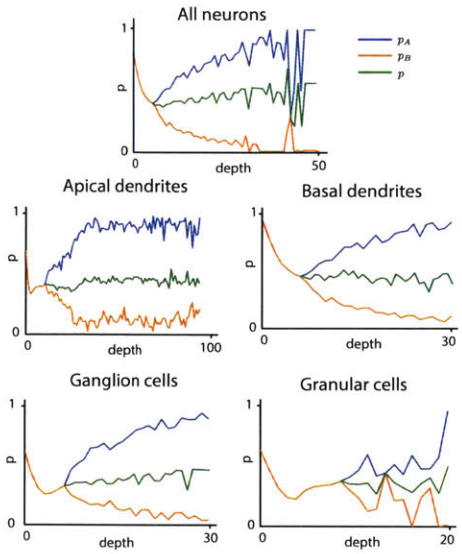


Figure 5-4: Fitted parameters for the depth-dependent symmetric and asymmetric models to dendrograms of different types of neurons. The x-axis measures distance from the soma.

2. Depth-dependent asymmetric models describe dendrograms very well, compared to models without asymmetry or depth-dependence. This suggests that branching does admit asymmetrical hidden variables and that distance from the soma plays a role in branching.

3. In fitting depth-dependent probabilities, we observe that models tend to be symmetric for low depth, with $p_A = p_B$, but then that these probabilities diverge strongly for greater depth. This cutoff is noteworthy, since it suggests that close to the soma, there is less differentiation of "main branches" and "side branches."

Chapter 6

Morphological Error Detection in 3D Segmentations

6.1 Introduction

The neural network of the brain remains a mystery, even as engineers have succeeded in building artificial neural networks that can solve a wide variety of problems. Understanding the brain at a deeper level could significantly impact both biology and artificial intelligence [12, 55, 90, 99, 148]. Perhaps appropriately, artificial neural networks are now being used to map biological neural networks. However, humans still outperform computer vision algorithms in segmenting brain tissue. Deep learning has not yet attained the intuition that allows humans to recognize and trace the fine, intermingled branches of neurons.

The field of *connectomics* aims to reconstruct three-dimensional networks of biological neurons from high-resolution microscope images. Automated segmentation is a necessity due to the quantities of data involved. In one recent study [57], the brain of a larval zebrafish was annotated by hand, requiring more than a year of human labor. It is estimated that mapping a single human brain would require a zettabyte (one billion terabytes) of image data [83], clearly more than can be manually segmented.

State-of-the-art algorithms apply a convolutional neural network (ConvNet) to predict, for each voxel of an image, whether it is on the boundary (*cell membrane*)

of a neuron. The predicted membranes are then filled in by subsequent algorithms [62]. Such methods are prone both to *split errors*, in which true objects are subdivided, and to *merge errors*, in which objects are fused together. The latter pose a particular challenge. Neurons are highly variable, unpredictably sprouting thousands of branches, so their correct shapes cannot be catalogued. Erroneously merged neurons are obvious to trained humans because they simply don't look right, but it has hitherto been impossible to make such determinations automatically.

We introduce a deep learning approach for detecting merge errors that leverages the morphological intuition of human annotators. Instead of relying upon voxelwise membrane predictions or microscope images, we zoom out and capture as much context as possible. Using only three-dimensional binary masks, our algorithm is able to learn to distinguish the shapes of plausible neurons from those that have been erroneously fused together.

We test our network, MergeNet, both on connectomics datasets and on an illustrative dataset derived from MNIST [81]. The key contributions of this approach include:

- **Localization of merge errors.** MergeNet is able to detect merge errors with high accuracy within a three-dimensional segmentation and to pinpoint their locations for correction (see Figures 6-1 and 6-2).

- **Unsupervised training.** The algorithm can be trained using any reasonably accurate segmentation, without the need for any additional annotation. It is even able to correct errors within its own training data.

- **Generalizability and scalability across datasets.** MergeNet can be applied irrespective of the segmentation algorithm or imaging method. It can be trained on one dataset and run on another with high performance. By downsampling volumetric data, our ConvNet is able to process three million voxels a second, faster than most membrane prediction systems.

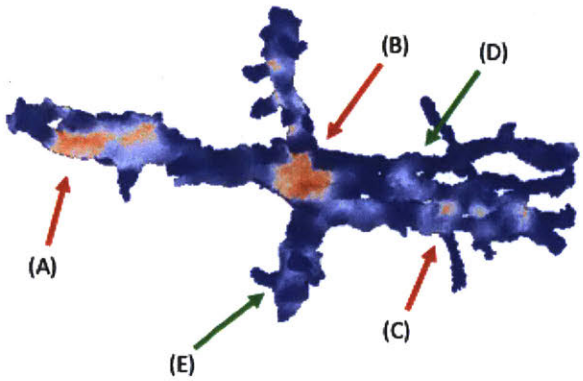


Figure 6-1: A probability map localizing merge errors, as predicted by MergeNet, for an object within the ECS dataset. Orange indicates a high probability of merge error, blue the absence of error. Location (A) illustrates a merge between two neurons running in parallel, (B) a merge between three neurons simultaneously (the two parallel neurons, plus a third perpendicular to them), and (C) a merge between a large neuron segment and a small branch from another neuron. MergeNet is able to learn that all of these diverse morphologies (and others not illustrated in this example) represent merge errors, but that locations such as (D) and (E) are normally occurring branch points within a single neuron.

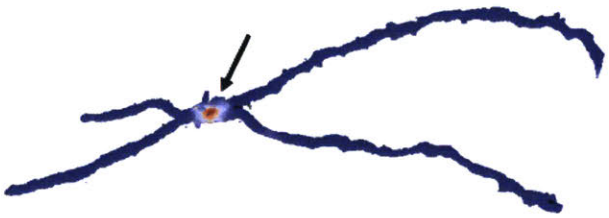


Figure 6-2: A single, relatively simple merge error detected and localized by MergeNet. This object, within the Kasthuri dataset [70], occurred *within the training data* of the algorithm, but was not labeled as a merge error. MergeNet was nonetheless able to correct the label. This capability allows MergeNet to be trained on an uncertain segmentation, then used to correct errors within the same segmentation, without requiring any manual annotation.

6.2 Related work

There have been numerous recent advances in using neural networks to recognize general three-dimensional objects. Methods include taking 2D projections of the input [143], combined 2D-3D approaches [20, 82], and purely 3D networks [92, 157]. Accelerated implementation techniques for 3D networks have been introduced by Budden et al. [18] and Zlateski, Lee and Seung [164].

Within the field of connectomics, Maitin-Shepard et al. [88] describe CELIS, a neural network approach for optimizing local features of a segmented image. Januszewski et al. [63] and Meirovitch et al. [94] present approaches for directly segmenting individual neurons from microscope images, without recourse to membrane prediction and agglomeration algorithms. Deep learning techniques have likewise been used to detect synapses between neurons [128, 132] and to localize voltage measurements in neural circuits [2] (progress towards a *functional connectome*). New forms of data are also being leveraged for connectomics [110, 145], thanks to advances in biochemical engineering.

Many authors cite the frequent problems posed by merge errors (see e.g. [107]); however, almost no approaches have been proposed for detecting them automatically. Meirovitch et al. [94] suggest a hard-coded heuristic to find "X-junctions", one variety of merge error, by analyzing graph theoretical representations of neurons as *skeletons* (see also [162]). Recent work including [73, 104] has considered the problem of deep learning on graphs, and Farhood, Ramkumar, and Kording [38] use Generative Adversarial Networks (GANs) to generate neuron skeletons. However, such methods have not to date been brought to bear on connectomic reconstruction of neural circuits.

6.3 Methods

Our algorithm, MergeNet, operates on an image segmentation to correct errors within it. Given an object within the proposed segmentation, MergeNet determines whether points chosen within the object are the location of erroneous merges. If no such points

exist, then the object is determined to be free from merge errors.

Input and architecture.

The input to our network is a three-dimensional *window* of the object in question, representing a $51 \times 51 \times 51$ section of the object, centered at the chosen point. (These dimensions are chosen as a tradeoff between enhancing speed and capturing more information, as we discuss further in sections §6.3.1 and 6.4.) Crucially, the window is given as a *binary mask*: that is, each voxel is 0 or 1 depending on whether it is assigned to the object. MergeNet is not given data from the original image, inducing the network to learn general morphological features present in the binary mask. The network follows a simple convolutional architecture, containing six convolutional layers with rectified linear unit (ReLU) activation, and three max-pooling layers, followed by a densely connected layer and softmax output. The desired output is a 1-hot vector for the two classes "merge" and "no merge", and is trained with cross-entropy loss.

2D MergeNet.

We also constructed a simpler 2D version of MergeNet to illustrate the identification of merge errors within two-dimensional images. In this case, the input to the network is a square binary mask, which passes through four convolutional layers and two max-pooling layers. This network was trained to recognize merges between binarized digits from the MNIST dataset [81]. Random digits were drawn from the training dataset and chained together, with merge errors given by pixels at the points of contact between neighboring digits. Testing was performed on similar merges created from the testing dataset. The size of the input window to the network was varied to compare accuracy across a variety of contextual scales.

Downsampling.

To apply MergeNet to connectomics data, we begin by downsampling all objects. Segmentations of neural data are typically performed at very high resolution, approximately 5 nm. The finest morphological details of neurons, however, are on the order of 100 nm. Commonly, data is anisotropic, with resolution in the z direction being significantly lower than that in x and y. We tested MergeNet with downsampling

ratios of $10 \times 10 \times 2$ and $25 \times 25 \times 5$ to compensate for anisotropy. Downsampling an object is performed by a max-pooling procedure. That is, every voxel within the downsampled image represents the intersection of the object with a corresponding subvolume of the original image with dimensions e.g. $25 \times 25 \times 5$.

Training.

The network was trained on artificially induced merge errors between objects within segmentations of neural tissue. Merges consisted of identifying immediately adjacent objects and designating points of overlap as the locations of merge errors. Negative examples consisted of windows centered at random points of objects within the segmentation. Artificial merge errors are used owing to the impracticality of manually annotating data over large enough volumes to determine sufficient merge errors for training. As we demonstrate, such training suffices for effective detection of real merge errors and has numerous other advantages (detailed in section §6.4). Training was performed on various segmentations of the Kasthuri dataset [70], and the algorithm was evaluated both on this dataset and on segmented objects of the ECS dataset, a $20 \times 20 \times 20$ micron cube of rat cortex data to which we were given access.

Output.

To run a trained instance of MergeNet on objects within a segmentation, it is not necessary to apply the network to every voxel since the predictions at nearby points may be interpolated. Sample points are therefore taken within each downsampled object, and MergeNet is run on windows centered at these points. The real-valued predictions at sample points are then interpolated over the entire object to give a heatmap of probabilities for merge errors. As the distribution of training examples is balanced between positive and negative examples, which is not true when the network is applied in practice, the output must be normalized or thresholded after the softmax layer. We find it effective to classify as a merge error any voxel at which the prediction exceeds 0.9.

6.3.1 Results

We first consider the illustrative example of the merged MNIST dataset. After training on five million examples within this dataset, we obtained a maximum pixelwise accuracy of **96.8 percent** on a test set constructed from held out digits and equally distributed between positive and negative examples. This corresponds to almost perfect identification of individual merge error regions, as shown in Figure 6-3. (Ambiguous pixels on the edges of merge error regions were the most likely pixels to be misclassified.)

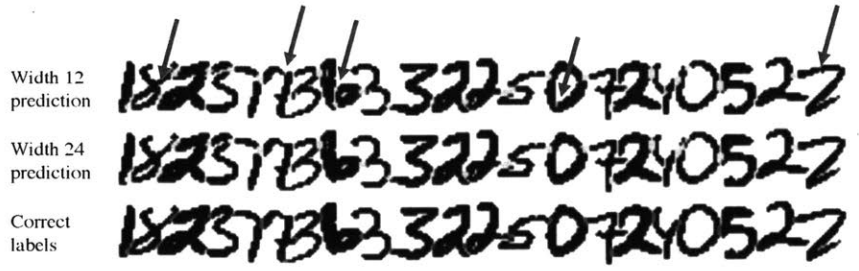


Figure 6-3: Predictions of MergeNet on merged MNIST digits, shown on a sample of the test set. The top image shows predictions of the network with 12×12 input windows, with predicted merges shown in yellow. The middle image shows predictions with 24×24 input windows. The final image shows the actual merges, in red. Note that both networks are quite accurate, but that for 12×12 input, the algorithm makes several erroneous predictions of merge errors (shown with blue arrows), which are not made for 24×24 input. This illustrates how greater morphological context leads to qualitatively better predictions. A quantitative assessment is shown in Figure 6-4.

Accuracy increases with morphological information.

In Figure 6-4, we show the dependence of accuracy on the window size used. Intuitively, a larger window gives the network more morphological context to work from, and plateaus in this case at approximately the dimensions of a pair of fused MNIST digits (40-50 pixels across), which represents the maximum scale at which morphology is useful to the network. Figure 6-3 provides a qualitative comparison of performance between a smaller and a larger window size. The smaller window size erroneously predicts merges within digits, while the large window size allows the network to recognize the shapes of these digits and identify only merge errors between digits.

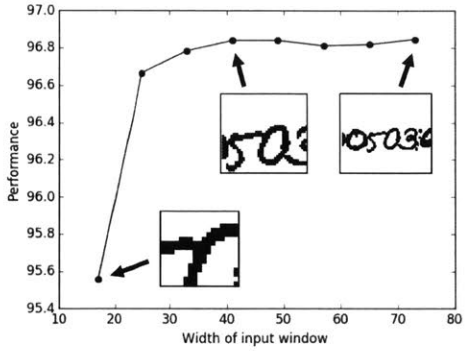


Figure 6-4: Performance of MergeNet on merged MNIST digits, shown as a function of input window width (with example windows shown). Note that increasing window size increases performance, but only up to a size of 40-50 pixels. We see that performance plateaus at the point at which morphological context captures all of the information about the neighboring digits. For the case of neurons, which are larger and more complicated, morphological context does not plateau; however, there is a tradeoff between more context and greater speed, since the time required to run the 3D ConvNet depends strongly upon input size.

There is, of course, a tradeoff between accuracy and the time required to train and run the network. Slowdown resulting from larger window size is considerable for three-dimensional ConvNets. We have attempted to choose the parameters of MergeNet with this tradeoff in mind.

MergeNet detects merge errors across datasets.
We trained MergeNet on two segmentations of the Kasthuri dataset [93] and tested performance on artificially merged objects omitted from the training set. Segmentation A was relatively poor, and training on it yielded performance of only **77.3 percent**. Segmentation B was more accurate, yielding performance of **90.0 percent**. Training on both segmentations together yielded the best performance on both test sets, showing that MergeNet is able to leverage contextual information from one segmentation to improve performance on another. We also note that after training on the low-quality Segmentation A, MergeNet was able to detect errors *within its own training set*, as shown in Figure 6-2.

MergeNet generalizes broadly across datasets as well as segmentations of the same dataset, and applies to both artificial and natural merge errors, though the latter are

	Training on Seg. A	Training on Seg. B	**Training on both**
Testing on Seg. A	77.3%	82.6%	**84.5%**
Testing on Seg. B	70.7%	90.0%	**91.9%**

harder to quantify owing to the paucity of large-scale annotation. After training on the Kasthuri dataset, MergeNet was able to detect naturally occurring merge errors within segmentations of the ECS dataset, obtained from a state-of-the-art U-Net segmentation algorithm [129]. Example output is shown in Figures 6-1 and 6-5.

6.4 Discussion

We will now consider the capabilities of the MergeNet algorithm and discuss opportunities that it offers within the field of connectomics.

Detection and localization of merge errors.
MergeNet is a powerful tool for detecting and pinpointing merge errors. Once a merge location has been flagged with high spatial precision, other algorithms can be used to create a more accurate local segmentation, thereby correcting any errors that occured. Flood-filling networks [63] and MaskExtend [94] are two examples of algorithms that have high accuracy, but are extremely time-consuming to run over large volumes, making them ideally suited to segment at the merge locations flagged by MergeNet. Alternatively, the agglomeration algorithm NeuroProof [106], used in transforming membrane probabilities to segmentations, can be tuned to be more or less sensitive to merge errors. A more merge-sensitive setting could be applied at those locations flagged by our algorithm.

If the thresholding step is omitted, then the output of MergeNet may be thought of as a probability distribution of merge errors over objects within a segmentation. This distribution may be treated as a Bayesian prior and updated if other information is available; multiple proofreading algorithms can work together. Thus, for example, synapse detection algorithms [128, 132] may provide additional evidence for a merge error if synapses of two kinds are found on the same segmented object, but are

normally found on different types of neurons. In such a scenario, the probability at the relevant location would be increased from its value computed by MergeNet, according to the confidence of the synapse detection algorithm. We envision MergeNet being the first step towards fully automated proofreading of connectomics data, which will become increasingly necessary as such data is processed at ever greater scale.

Unsupervised training.

MergeNet is trained on merge errors created by fusing adjacent objects within a segmentation. This allows training to proceed without any direct human annotation. In testing MergeNet, we performed training on several automatically generated segmentations of EM data and obtained good results, even though the training data was not free of merge errors and other mistakes. It is highly advantageous to eliminate the need for further data annotation, since this is the step in connectomics that has traditionally consumed by far the most human effort. The ability to run on any (reasonably accurate) segmentation also means that MergeNet can be trained on far larger datasets than those for which manual annotation could reasonably be obtained. Automated segmentations already exist for volumes of neural tissue as large as 232,000 cubic microns [114].

Comparison of segmentations.

Automatic detection of merge errors allows us to compare the performance of alternative segmentation algorithms *in the absence of ground truth annotation.* We ran MergeNet with the same parameter settings on two segmentations within the ECS dataset, after training on other data. These alternative segmentations were produced by two different versions of a state-of-the-art U-Net segmentation algorithm [129]. For the simpler algorithm, 33 of the 300 largest objects within the segmentation (those most likely to have merge errors) were flagged as unlikely, while, for the more advanced algorithm, only 15 of the largest 300 objects were flagged. This indicates that the latter pipeline produces more plausible objects, making fewer merge errors. The size of the objects was comparable in each case, so there is no indication that this improvement came with a greater propensity for erroneous splits within single objects. Thus, MergeNet was able to perform a fully automatic comparison of two

segmentation algorithms and confirm that one outperforms the other.

Correction of the training set.

We have already observed that MergeNet can be trained on any reasonably correct segmentation. In fact, it is possible to leverage artificial merge errors within the training set to detect real merge errors that may occur there (as shown in Figure 6-2). This is remarkable, since the predictions MergeNet makes in this case should conflict with the labels it was itself trained on - namely, when objects from the training set that have not been artificially merged are nonetheless the result of real merge errors. Our observations align with results showing that neural networks are capable of learning from data even when the labels given are unreliable [144].

Independence from lower-order errors.

Since MergeNet makes use of the global morphology of neurons, it is not reliant on earlier stages of the connectomics pipeline, such as microscope images or membrane predictions. Thus, it is able to correct errors that arise at early stages of the pipeline, including those at the experimental stage. EM images are prone to various catastrophic errors; most notably, individual tissue slices can tear before imaging, leading to distortion or gaps in the predicted membranes. Algorithms that stitch together adjacent microscope images also sometimes fail, leading to pieces of neurons in one image being erroneously aligned with pieces from the neighboring image. Typically, such errors are propagated or magnified by later stages of the connectomics pipeline, since algorithms such as watershed and NeuroProof [106] assume that their input is mostly true. By contrast, MergeNet can look at the broader picture and use "common sense" as would a human proofreader.

The output of a membrane-detection algorithm also can induce errors in object morphology. Common sources of error at this stage are ambiguously stained tissue slices and intracellular membranes, such as those from mitochondria, which can be confused with the external cell membrane. Figure 6-5 shows an error in predictions from the U-Net algorithm, where a large gap in the predicted membrane has allowed two objects to be fused together into one (shown in blue). By utilizing the overall 3D context, MergeNet is able to detect and localize the error (shown in red).

Generalizability to different datasets.

One of the challenges of traditional connectomics algorithms is that there are numerous different imaging techniques, which can each be applied to the nervous systems of various organisms, and in some cases also to structurally distinct regions of the nervous system within a single organism. For networks used in connectomics for image segmentation, it is often necessary to obtain ground truth annotations on each new dataset, which consumes considerable time and effort.

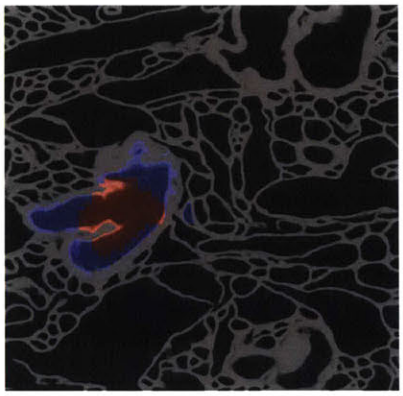


Figure 6-5: The output of MergeNet at a merge error, superimposed over the erroneously predicted membranes that led to the merge error. MergeNet output at individual pixels has been thresholded above 0.9, with red denoting predicted merge error and blue the absence of error. Observe that the predicted membranes have a wide gap at the region MergeNet has flagged; this gap is incorrect and the membrane should extend between the two objects, separating them. Note that MergeNet used only three-dimensional morphological information to detect this error, and did not make use of the (erroneous) membrane predictions that are shown, or the underlying microscope images.

MergeNet, by contrast, is highly transferrable between datasets. Not only can the algorithm be trained on an unverified segmentation and can correct it, but it can also be trained on one dataset and then run on a segmentation of a different dataset, without any retraining. Figures 6-1 and 6-5 show images obtained by training on a segmentation of the Kasthuri dataset [70] and then running on the ECS dataset.

Applicability to anisotropic data.

The microscope data underlying connectomics segmentation is often anisotropic, where the particular dimensions in the x, y, z directions depend upon the particu-

Figure 6-6: Glial cell, flagged as a merge error by MergeNet. While glia are not merge errors, they are also not neurons and did not occur in the training set for MergeNet. As the algorithm recognizes, the morphology of glia is markedly different from that of neurons. Specifically training MergeNet to recognize glia could be useful in segmenting these cells, which occur along with neurons in brain tissue.

lar imaging procedure used. For example, the Kasthuri dataset has resolution of $6 \times 6 \times 30$ nanometers, while our ECS dataset is even more anisotropic, with resolution of $4 \times 4 \times 30$ nanometers. Some imaging technologies do yield isotropic data, such as expansion microscopy (ExM) [19] and focused ion beam scanning electron microscopy (FIB-SEM) [75]. Various techniques have been proposed to work with anisotropic data, including 2D ConvNets feeding into 3D ConvNets [82] and a combination of convolutional and recurrent networks [20].

MergeNet cancels the effect of anisotropy, as necessary, by downsampling differentially along the x, y, z directions. Thus, the network is able to transform any segmentation into one in which morphology is approximately isotropic, making learning much easier. We also anticipate that it may be possible to train on data from one imaging modality, then to apply a different downsampling ratio to run on data with different anisotropy. For example, MergeNet could be trained on an EM segmentation, then run on an ExM segmentation.

Scalability.

The MergeNet algorithm is designed to be scalable, so that it can be used to proofread segmentations of extremely large datasets. The network is applied only once objects have been downsampled by a large factor in each dimension, and is applied then only

to sampled points within the downsampled object. These two reductions in cost allow the 3D ConvNet to be run at scale, even though 3D kernels are slower to implement than 2D kernels.

We tested the speed of MergeNet on an object with 36,874,153 original voxels, downsampled to 18,777 voxels, from which we sampled 1,024, allowing us to generate a dense probability map across the entire object. The ConvNet ran in 11.3 seconds on an Nvidia Tesla K20m GPU. This corresponds to a speed of over three million voxels per second within the original image. Thus, the network could be applied to a volume of 1 billion voxels in a minute using five GPUs. By comparison, the fastest membrane-prediction algorithm can process 1 billion voxels within 2 minutes on a 72-core machine [93], demonstrating that our algorithm can be integrated into a scalable connectomics pipeline. Note that our experiments were performed using TensorFlow [1]; we have not attempted to optimize time for training or running the network, though recent work indicates that significant further speedup may be possible [18, 164].

Detection of non-neuronal objects.

While MergeNet is trained only on merge errors, it also seems to be able to detect non-neuronal objects, as a byproduct of learning plausible shapes for neurons. In particular, we observe that MergeNet often detects glia (nonneuronal cells that occur in neural tissue), the morphologies of which are distinctively different from those of neurons. Figure 6-6 shows an example of a glial object from the ECS dataset; notice that MergeNet finds the morphology implausible, even though it has been trained on neither positive nor negative examples of glia. Quantifying the accuracy of glia detection is challenging, however, since little ground truth has been annotated for this task, and most connectomics algorithms are unable to distinguish glia.

Finally, let us consider when (and why) MergeNet succeeds and fails on different inputs. The algorithm does not simply label all branch points within a neuron as merge errors, or else it would be effectively useless. However, the network can be confused by examples such as two branches that diverge from a main segment at approximately the same point, resembling the cross of two distinct objects. MergeNet

also misses some merge errors. For example, when two neuronal segments run closely in parallel, there may at some points along the boundary be no morphological clues that two objects are present. It is worth noting, however, that parallel neuronal segments can in fact be detected by MergeNet, as shown at point (A) of Figure 6-1.

6.5 Conclusion

Though merge errors occur universally in automated segmentations of neural tissue, they have never been addressed in generality, as they are difficult to detect using existing connectomics methods. We have shown that a 3D ConvNet can proofread a segmented image for merge errors by "zooming out", ignoring the image itself, and instead leveraging the general morphological characteristics of neurons. We have demonstrated that our algorithm, MergeNet, is able to generalize without retraining to detect errors within a range of segmentations and across a range of datasets. Relying solely upon unsupervised training, it can nonetheless detect errors within its own training set. Our algorithm enables automatic comparison of segmentation methods, and can be integrated at scale into existing pipelines without the requirement of additional annotation, opening up the possibility of fully automated error detection and correction within neural circuits.

While MergeNet can detect and localize merge errors, it cannot, by itself, correct them. One could conceive a variation on the MergeNet algorithm that is used to mark the exact boundary of a merge error, allowing a cut to be made automatically along the boundary so as to correct the merge without additional effort. However, in practice this is a much more challenging task. Often it is impossible to determine the exact division of objects at a merge error purely from morphology. For instance, when two largely parallel objects touch, it may not be evident which is the continuation of which past the point of contact, even if the erroneous merge itself is obvious. Likewise, some merge errors consist of three or more objects that have been confused in some complex way, e.g. by virtue of poor image quality at that location. In such cases, any merge-correction algorithm must have recourse to the underlying microscope images

or membrane probabilities, rather than relying purely upon morphological cues.

Deep learning approaches leveraging morphology have the potential to transform biological image analysis. It may, for instance, become possible to classify types of neurons automatically, or to identify anomalies such as cancer cells. We anticipate a growth in such algorithms as the scale of biological data grows and as progress in connectomics leads to a deeper understanding of the brain.

Chapter 7

Modeling Characteristics of Biological Attractor Networks

7.1 Markov Transitions between Attractor States in a Recurrent Neural Network

7.1.1 Motivation

With the advancement of probabilistic theories of human cognition [45], there has been increasing interest in neural mechanisms that can represent and compute these probabilities. Several new models of neural computation carry out Bayesian probabilistic inference taking into account both data and prior knowledge, and can represent uncertainty about the conclusions they draw [87, 109, 139]. In many tasks, neural mechanisms are required that can transition stochastically to a new state depending on the current state: for example, to predict the path of a moving object [154], gauge the effect of a collision [131], or estimate the dynamic motion of fluids [8], as well as in the general context of carrying out correlated sampling over a posterior distribution [42, 16, 36]. The Markov transition probabilities in these cases are dictated by knowledge of the world. The stochasticity of transitions allows decisions that are tempered by uncertainty, rather than making a "best guess" or point estimate that is agnostic

to uncertainty and is chosen deterministically based on some measure of optimality. Further, Markov chain Monte Carlo methods [103] allow us to engineer a Markov chain with stationary distribution equal to any distribution of interest. Therefore a simple Markov chain with the right transition probabilities can also form the basis for neurally plausible probabilistic inference on a discrete state space.

It is important here to distinguish between stochasticity in our perception or neural representation of states, and stochasticity incorporated into a computational step. The first is unavoidable and due to noise in our sensory modalities and communication channels. The second is inherent to a process the brain is carrying out in order to make probabilistic judgments, and represents useful information about the structure of the environment. While it is difficult to tease apart these sources of noise and variability, Beck et al. [9] suggest that sensory or representational noise is not the primary reason for trial-to-trial variability seen in human responses and that there are other sources of stochasticity arising from the process of inference that might be more important and influential in explaining observed behavioral variability. Humans are in fact remarkably immune to noise in percepts - for example when identifying occluded objects and filtering out one source of sound amid ambient noise [47, 67].

Hopfield networks represent an effective model for storage and representation that is largely immune to noise; different noisy or partial sensory percepts all converge to the same memory as long as they fall within that memory's basin of attraction. These "memory" states are represented in a distributed system and are robust to the death of individual neurons. Stochastic transitions in Hopfield networks therefore are a step towards stochastic computation that still ensures a noise-robust representation of states.

The Markov chain dynamics we model also have applications in systems where experimental verification is more lucid. For example, the Bengalese finch's song has been effectively modeled as a hidden Markov model [65]. While deterministic birdsong in the zebra finch has previously been modeled by feedforward chains of neurons in HVC [85], our network provides a potential neural model for stochastic birdsong. Further, its specific structure has possible parallels in songbird neural architecture,

as we later detail.

7.1.2 Background

A Hopfield network [59] is a network of binary neurons with recurrent connections given by a symmetric synaptic weight matrix, J_{ij}. The state x_i of the ith neuron is updated according to the following rule:

$$x_i \leftarrow \text{sign}\left(\sum_{j=1}^{n} J_{ij} x_j\right) \tag{7.1}$$

With this update rule, every initial state of the network deterministically falls into one of a number of stable fixed points which are preserved under updates. The identity of these fixed points (*attractors* or *memories*) can be controlled by appropriate choice of J_{ij}, according to any of various learning rules [54, 130, 142, 58]. If the network is initialized at a corrupted version of a memory, it is then able to converge to the true memory, provided that the corrupted/noisy initialization falls within the true memory's basin of attraction. This allows Hopfield networks to be a model for *content-addressable, associative* memory.

Due to symmetry of weights, a traditional Hopfield network always converges to a stable attractor state. By adding asymmetric connections, it is possible to induce transitions between the attractor states. Sompolinsky and Kanter [140] show that a set of *deterministic* transitions between attractor states can be learned with a Hebbian learning rule, by means of time-delayed *slow connections*. Here, the transition structure is built into the synapses of the network and is not stochastic. The challenge we address in this paper is to leverage what we know from past work about deterministic transitions in attractor networks and combine it with a source of noise to make these transitions stochastic, with controllable Markov probabilities for each transition.

7.1.3 Network architecture

We propose a network consisting of three parts: A *memory network*, a *noise network*, and a *mixed network* (see Fig. 7-1). The memory network taken by itself is an attractor network with stabilizing recurrent connections; it stores states of the Markov chain as attractors. The noise network also stores a number of attractor states (the *noise states*); in its case, the transitions between attractors occur uniformly at random.

The mixed network is another attractor network, which receives input from both the memory and noise networks, according to fixed random weights. The attractors (*mixed states*) of the mixed network are chosen according to the memory and noise states; thus, a different pair of memory state and noise state will induce the mixed network to fall into a different attractor. The memory network receives input from the mixed network, which induces it to transition between the memory attractor states.

The key insight in our design is that given the combined state of the noise and memory networks (as captured in the mixed network), the next memory state is fully determined. Stochasticity arises from resampling the noise network and allowing it to fall uniformly at random into a new attractor. This is in fact the sole source of stochasticity in the model, and it is in a sense analogous to the reparameterization trick used in Kingma and Welling [72].

In order for transitions between memory states to be determined by the state of mixed network, the attractors for the mixed network should be linearly separable. A simple concatenation of memory and noise states would result in a strong linear dependence between mixed states, making them difficult to linearly separate [26]. We recover linear separability in our model by instead constructing the mixed network as a random projection of memory and noise states into a higher dimensional space [6].

The connections from the mixed network back to the memory network that induce the transition are *slow connections* (see [140]); they are time-delayed by a constant τ and are active at intervals of τ. This allows the memory network to stabilize its previous state before a transition occurs. Thus, at every time step, each memory neuron takes a time-delayed linear readout from the mixed representation, adds it to

the Hopfield contribution from the memory network and passes the sum through a threshold non-linearity.

Formally, the dynamics are given by the following equations, where x_i^M $(0 \leq i < n_M)$, x_j^N $(0 \leq j < n_N)$, and x_k^Q $(0 \leq k < n_Q)$ denote states of neurons in the memory network, noise network, and mixed network, respectively. The function $\delta_\tau^{\text{mod}}(t)$ is 1 when $t \equiv 0 \pmod{\tau}$, otherwise 0; and the notation $x(t-\tau)$ denotes the state x at time $t-\tau$ (otherwise assumed to be time t). The function $\nu(t,\tau)$ represents a noise function that is resampled uniformly at random at intervals of τ.[1]

$$
\begin{aligned}
x_i^M &\leftarrow \text{sign}\left(\sum_{\ell=1}^{n_M} J_{i\ell}^M x_\ell^M + \delta_\tau^{\text{mod}}(t) \sum_{k=1}^{n_Q} J_{ik}^{MQ} x_k^Q(t-\tau)\right), \\
x_j^N &\leftarrow \text{sign}\left(\sum_{\ell=1}^{n_N} J_{j\ell}^N x_\ell^N + \nu(t,\tau)\right), \\
x_k^Q &\leftarrow \text{sign}\left(\sum_{\ell=1}^{n_Q} J_{k\ell}^Q x_\ell^Q + \sum_{i=1}^{n_M} J_{ki}^{QM} x_i^M + \sum_{j=1}^{n_N} J_{kj}^{QN} x_j^N\right),
\end{aligned}
$$

The weight matrices J^M, J^N, J^Q, and J^{MQ} are learned (see below), while J^{QM} and J^{QN} are random, with $n_Q \gg n_M, n_N$.

We implement the noise network as a *ring attractor* — a ring of neurons where activating any contiguous half-ring yields an attractor state. Here we have adapted the model described in Ben-Yishai et al. [11] to the discrete setting according to the following dynamics:

$$
J_{ij}^N = \begin{cases} -1 & \text{for } n_N/4 \leq |i-j| \leq 3n_N/4, \\ +1 & \text{otherwise,} \end{cases}
$$

for $1 \leq i, j \leq n_N$, where we require that n_N be even. There are, then, $a_N = n_N$

[1] This "clocked" activity, while not biologically implausible, might be unnecessary; future work might aim to replace it, perhaps by combining time-delayed neurons with a *sparse*, high-dimensional projection to the mixed representation.

attractor states A_i, which take the form:

$$A_i = \begin{cases} x^N_{i+k} = -1 & \text{for } 0 \leq k < n_N/2, \\ x^N_{i+k} = 1 & \text{for } n_N/2 \leq k < n_N, \end{cases}$$

where indices are taken modulo n_N.

Another possible construction for the noise network is simply to have a small set of randomly activated neurons with no recurrent stabilizing connections. In this construction, the number of noise attractor states is exponential in the number of noise neurons, allowing higher precision in probabilities for the same number of noise neurons. However, this construction has the disadvantage that it is highly sensitive to the perturbation of single neurons, and so it may be difficult to distinguish between incidental noise and a resampling of the noise network.

The components in our network all have biological analogues. We use slow neurons to prompt transitions between memories only after the memories have been allowed to stabilize. These could be implemented via the *autapses* in [136]. We use large random expansions to increase linear separability of states, as suggested in [5]. Also, the noise network in our architecture has a promising parallel in the LMAN region of the songbird brain, which has been linked to generating variability in songs during learning [105]. Alternatively, the noise network in our model could just be any uncorrelated brain region.

7.1.4 Learning

There are several sets of weights that must be determined within our model. Those denoted J^M, J^N, J^Q, and J^{MQ} above are learned, while J^{QM} and J^{QN} are random.

The recurrent connections within the memory network (J^M) and noise network (J^N) are learned using Hebb's rule, ensuring that each of the three subnetworks has the desired attractor states. The weights for slow synapses from the mixed network to the memory network (J^{MQ}) are chosen according to methods described in [140] for inducing deterministic sequences of attractors. The weights within the mixed network

(J^Q) are learned using the perceptron learning rule, yielding a larger capacity than the Hebbian approach used in [140].

Finally, it is necessary that we determine which (`memory`, `noise`) state pairs should transition to which new memories. For a desired transition $S_1 \rightarrow S_2$ between memory states, having probability p in the Markov chain, we assign (approximately) a p-fraction of noise states, so that (S_1, N) induces a transition to S_2 for all N in the p-fraction. Thus, several noise states, in the presence of a particular memory state, could result in the same transition; the number of noise states assigned to a transition is proportional to the probability of that transition. Probabilities may be approximated to within an accuracy of $\epsilon = O(1/a_N)$, where a_N is the number of attractor states in the noise network.

Let us consider the biological feasibility of our learning approach. We use a Hebbian rule for stabilizing the patterns, in keeping with the established hypothesis of Hebbian learning within the brain. For learning state transitions, we have so far opted to use the perceptron rule in the interest of increasing capacity. This, however, requires storing and iterating over a list of pairs of the form (`memory`, `noise`). It is perhaps more biologically feasible to use an online algorithm for which such a list need not be learned and stored, where learning proceeds by sampling trajectories from the desired Markov chain.

Such an online learning rule indeed seems possible if we use Hebbian learning for the state transitions. Specifically, every time a transition occurs in the real world, we strengthen the corresponding connections in our network. Since our transitions take the form of (`memory` + `noise`) $\rightarrow$ `new memory`, the noise state used in the transition is simply whatever (arbitrary) state the noise network is in at that moment. A slight weakness to this approach is the network must recognize when a particular transition had already been learned, to prevent overwriting; however, this seems by no means biologically insurmountable.

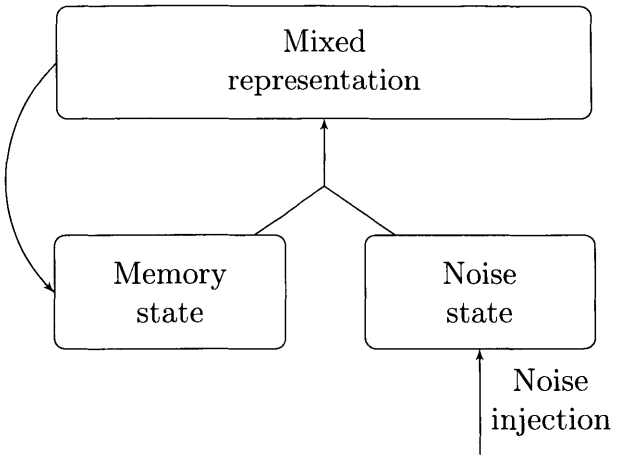


Figure 7-1: A schematic of our recurrent network. The current memory state is paired with a randomized noise state, and that pairing determines the next memory state. Since the noise state is sampled uniformly at random, it follows that the probability of a particular transition from memory state S_1 to S_2 is given by the fraction of noise states that pair with S_1 to produce S_2. To implement these transitions, we consider each memory neuron at time $t+1$ to be a perceptron readout of the mixed representation. To increase the separability ability of these perceptrons, the mixed representation is a large, random expansion of the (`memory`, `noise`) pairings. The state transitions operate on a slow timescale due to the slow neurons. Not pictured are the self-connections within the memory network, noise network, and mixed network that serve to stabilize the corresponding states on a fast timescale.

7.1.5 Future directions

The connection weights from the mixed network to the memory network must discriminate between the various mixed states in order to elicit the right transition. We use a perceptron learning rule to learn these weights and use a high-dimensional random projection to form our mixed state, to increase the number of mixed states that are linearly separable [6, 26], thereby allowing us to encode a larger number of transitions into our network. In effect, points that were close together in the original space are far apart in the higher dimensional space, and thus easy to separate. This comes with a downside, however, since small, accidental perturbations in the original state will likewise be blown up by the random expansion. In Barak et al. [6], this problem is referred to as a generalization-discrimination trade off.

We did not encounter this problem in our preliminary simulations, since we did not consider noise in our neural update rule, Equation 7.1. This ensured that our

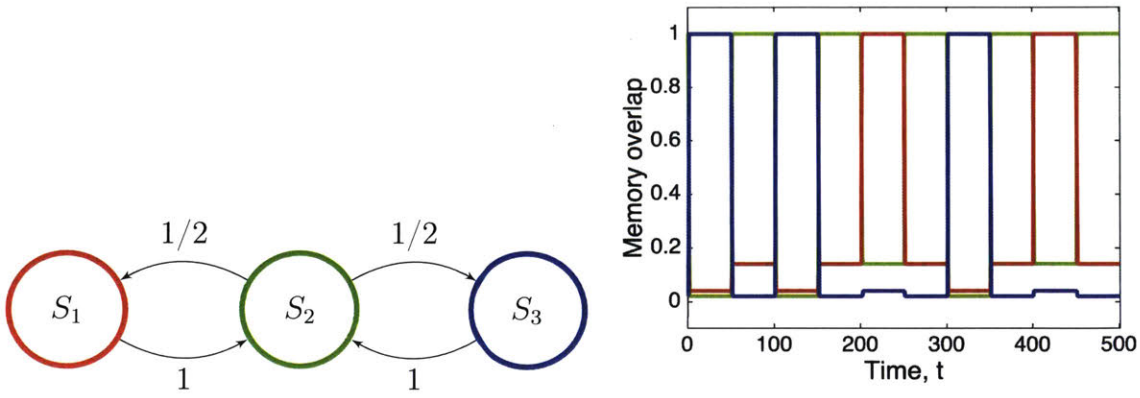


Figure 7-2: Top: a simple Markov chain, where the dynamics ensure the central state (green) is visited on alternate time steps with probability 1. The number of times state S1 is visited is therefore a binomial random variable $\sim B(n, p = 0.5)$. Bottom: results of our network simulating the dynamics of this Markov chain. We associate a distinct 'memory vector' with each Markov state S_i, and the plot shows the overlap of each such memory vector with the system state at every time step of the simulation. Averaging over 4000 state transitions, our simulation yielded empirical transition probabilities of $P(S_1|S_2) = 0.512 \approx P(S_3|S_2) = 0.488$. These values are within one standard deviation of the mean of the binomial distribution $\sim B(n = 2000, p = 0.5)$, indicating that our recurrent neural network is operating correctly. Note that we use $n = 2000$ instead of 4000 because half of the transitions go deterministically to state S_2.

system always fell to the very bottom of its attractor states, and in the statistical mechanics analogy corresponds to operating the system at zero temperature.

We plan to test empirically the limits of our system's performance with respect to the level of noise in our update rule (the temperature of our system), the expansion ratio of our random projection, and the number of mixed states that need to be correctly classified. Checking the agreement of our simulations with theoretical predictions will reveal more about the capabilities of a system of neural computations based on stochastic transitions between attractor states.

Our architecture can also implement Markov chain Monte Carlo, specifically some version of Gibbs sampling. Each new state is sampled from a distribution conditioned on the current state - this is analogous to sampling from a conditional distribution as in Gibbs sampling. Future work could involve learning and representing these conditionals within our network and implementing a noise-robust stochastic sampler

over discrete state spaces.

7.2 Periodic Trajectories in Threshold-Linear Networks

Attractor states in networks need not be fixed points of the network dynamics, as in Hopfield-type models. Central pattern generators in the brain are believed to demonstrate attractor states that follow periodic behavior [159]; that is, a certain pattern of neurons fires sequentially, and then the sequence repeats. Such repetitive sequences underlie many behaviors such as locomotion. In this section, we model such behavior using threshold-linear networks.

A *threshold-linear network* is a dynamical system of the form:

$$\frac{d\boldsymbol{x}}{dt} = -\boldsymbol{x} + [W\boldsymbol{x} + \boldsymbol{b}]_+, \tag{7.2}$$

for $\boldsymbol{x} \in \mathbb{R}^n$, where $W \in \mathbb{R}^{n\times n}$ is a constant matrix, $\boldsymbol{b} \in \mathbb{R}^n$ is a constant vector, and $[\bullet]_+$ denotes $\max(\bullet, 0)$. Such a system may be conceived as a continuous analogue of the Hopfield model, where the coordinates of the vector $\boldsymbol{x}$ represent the activities of neurons in the system. Neuron activities experience natural decay under this model (the $-x$ term above), but are subject to a constant stimulation $\boldsymbol{b}$ and are also affected by the activities of other neurons according to the connectivity matrix W.

The *combinatorial threshold-linear network (CTLN)* framework allows us to isolate the connectivity component of these dynamics by setting the parameters of the system purely according to a graph representing connectivity between the neurons. Suppose that constants ϵ, δ, θ satisfy $\delta, \theta > 0$, and let G be a directed graph on vertices $v_1, v_2, \ldots, v_n$. Then, the CTLN defined by $\epsilon, \delta, \theta, G$ is the threshold-linear network $\boldsymbol{x}$

for which the dynamics (7.2) above satisfy $\boldsymbol{b}_i = \theta$ for all i, and

$$W_{ij} = \begin{cases} 0 & \text{if } i = j \\ -1+\epsilon & \text{if } i \neq j \text{ and } v_i \leftarrow v_j \\ -1-\delta & \text{if } i \neq j \text{ and } v_i \not\leftarrow v_j \end{cases}$$

for all i, j. From a neuroscientific standpoint, this model captures a scenario in which neuron activities are subject to a constant excitation θ, weak inhibition from neighboring neurons, and strong inhibition from non-neighboring neurons.

Thus, for example, in the case of G a directed 3-cycle, the dynamics $\boldsymbol{x}$ are given by:

$$\begin{aligned} \frac{d\boldsymbol{x}}{dt} &= -\boldsymbol{x} + [W\boldsymbol{x} + \boldsymbol{b}]_+ \\ \boldsymbol{b} &= (\theta, \theta, \theta) \\ W &= \begin{pmatrix} 0 & -1-\delta & -1+\epsilon \\ -1+\epsilon & 0 & -1-\delta \\ -1-\delta & -1+\epsilon & 0 \end{pmatrix}. \end{aligned}$$

Morrison et al. [100] and Curto, Geneson, and Morrison [27] have considered the emergent behavior of such networks (see also [28]), observing that even the CTLN simplification captures complex behavior observed within the general threshold-linear model. Morrison et al. [100] observed that stable periodic trajectories seem to arise in CTLNs on many graphs, notably including directed cycles, in the regime $0 < \epsilon < \delta/(\delta+1)$. (Various graphs also have been observed to give rise to other notable behavior, such as quasiperiodic orbits that trace out surfaces of positive codimension.) However, proving the existence of periodic trajectories has proved challenging even for simple graphs. We here provide the first and only such result, for the directed 3-cycle.

Theorem 7.2.1. *The dynamics $\boldsymbol{x}(t)$ above for G a directed 3-cycle possess a periodic trajectory, for $0 < \epsilon < \delta/(\delta+1)$.*

In order to prove the theorem, we will restrict ourselves to considering the cube R defined by $0 \leq x_1, x_2, x_3 \leq \theta$.

Lemma 7.2.2. *If $\boldsymbol{x}(0) \in R$, then $\boldsymbol{x}(t)$ remains within R for all t.*

Proof of Lemma. Suppose first that $x_i(t) = 0$ (for some time t and some $i \in \{1, 2, 3\}$). Then, $(d\boldsymbol{x}/dt)_i = [W_{i\bullet}\boldsymbol{x} + \theta]_+ \geq 0$. Likewise, if $x_i(t) = \theta$, then $(d\boldsymbol{x}/dt)_i = -\theta + [W_{i\bullet}\boldsymbol{x} + \theta]_+ \leq -\theta + \theta = 0$.

Hence, $\boldsymbol{x}(t)$ never leaves R for all t. □

Let H_i, for $i = 1, 2, 3$, denote the plane $(W\boldsymbol{x} + \boldsymbol{\theta})_i = 0$. Let H_i^+ and H_i^- denote respectively the regions $(W\boldsymbol{x} + \boldsymbol{\theta})_i \geq 0$ and $(W\boldsymbol{x} + \boldsymbol{\theta})_i \leq 0$. Define the following regions:

$$
\begin{aligned}
R_\emptyset &= R \cap H_1^+ \cap H_2^+ \cap H_3^+ \\
R_1 &= R \cap H_1^- \cap H_2^+ \cap H_3^+ \\
R_{12} &= R \cap H_1^- \cap H_2^- \cap H_3^+ \\
R_{123} &= R \cap H_1^- \cap H_2^- \cap H_3^-,
\end{aligned}
$$

and define R_2, R_3, R_{23}, R_{31} similarly. These regions partition R.

Let $r(t)$ denote the distance between $\boldsymbol{x}(t)$ and the line $c \cdot \langle 1, 1, 1 \rangle$, and let $z(t)$ denote the length of the projection of $\boldsymbol{x}(t)$ onto $\langle 1, 1, 1 \rangle$.

Lemma 7.2.3. *Within the region $R_\emptyset$, we have $dr/dt = (\delta - \epsilon)r(t)/2$ and $dz/dt = (-3 - \delta + \epsilon)z(t) + \sqrt{3}$.*

Proof of Lemma. Within $R_\emptyset$, the behavior of $\boldsymbol{x}$ is governed by $d\boldsymbol{x}/dt = (W - I)\boldsymbol{x} + \boldsymbol{\theta}$. Note that $\frac{\sqrt{3}}{3}\langle 1, 1, 1 \rangle$ is the only real eigenvector of $W - I$, corresponding to the eigenvalue $-3 - \delta + \epsilon$. This shows that $dz/dt = (-3 - \delta + \epsilon)z(t) + \sqrt{3}$.

Let $v(t)$, $w(t)$ respectively denote the components of $\boldsymbol{x}(t)$ in the directions of the orthonormal vectors $\frac{\sqrt{2}}{2}\langle 1, -1, 0 \rangle$ and $\frac{\sqrt{6}}{6}\langle 1, 1, -2 \rangle$. Observe that $W - I$ acts on these

vectors according to the matrix

$$W' = \begin{pmatrix} (\delta-\epsilon)/2 & \sqrt{3}(\delta+\epsilon)/2 \\ -\sqrt{3}(\delta+\epsilon)/2 & (\delta-\epsilon)/2 \end{pmatrix}.$$

Therefore, we have:

$$\begin{aligned}
\frac{dr}{dt} &= \frac{d}{dt}\sqrt{v^2(t)+w^2(t)} \\
&= \left(v(t)\frac{dv}{dt} + w(t)\frac{dw}{dt}\right)\left(\frac{1}{\sqrt{v^2(t)+w^2(t)}}\right) \\
&= \left(v(t),w(t)\right)^T W'(v(t),w(t))\left(\frac{1}{\sqrt{v^2(t)+w^2(t)}}\right) \\
&= \left(\frac{\delta-\epsilon}{2}\left(v(t)^2+w(t)^2\right)\right)\left(\frac{1}{\sqrt{v^2(t)+w^2(t)}}\right) \\
&= \left(\frac{\delta-\epsilon}{2}\right) r(t).
\end{aligned}$$

□

From Lemma 7.2.3, we see that $\boldsymbol{x}(t) = \frac{1}{3+\delta-\epsilon}\langle 1,1,1\rangle$ is a fixed point, which we denote $\boldsymbol{y}$. Let S_ρ be the open ball of radius ρ about $\boldsymbol{y}$, with ρ small enough that S_ρ is contained wholly in $R_\emptyset$. Let Q_ρ be the set of $\boldsymbol{x}(0) \in R$ such that $\boldsymbol{x}(t) \in S_\rho$ for some t. From Lemma 7.2.3 it follows that all points in $Q_\rho \cap R_\emptyset$ are at less than distance ρ from the line $x_1 = x_2 = x_3$.

Note that we have $d\boldsymbol{x}/dt = -\boldsymbol{x}$ for all points in R_{123} such that $x_1 = x_2 = x_3$. Since $d\boldsymbol{x}/dt$ is continuous in R, we may conclude that for each ρ', there exists ρ such that all points in P_ρ are at less than distance ρ' from the line $x_1 = x_2 = x_3$. We are thus able to pick ρ small enough that (i) S_ρ is contained in $R_\emptyset$, and (ii) $R\backslash Q_\rho$ is homeomorphic to a torus.

Let $R' = R\backslash Q_\rho$, let $R'_\emptyset = R_\emptyset\backslash Q_\rho$, and define $R'_1, R'_2, R'_3, R'_{23}, R'_{13}, R'_{12}, R'_{123}$ similarly. It is simple to verify that regions $R_\emptyset, R_2, R_3, R_{23}$ intersect in the points $\left\langle \frac{1}{1+\delta}, 0, \frac{\epsilon+\delta}{(1+\delta)^2}\right\rangle$ and $\frac{1}{2+\delta-\epsilon}\langle 1,1,1\rangle$. The plane passing through these points and the origin intersects R' in two disconnected surfaces. Let A_1 be the surface which is on

the same side of the line $\langle t, t, t \rangle$ as the point $\left\langle \frac{1}{1+\delta}, 0, \frac{\epsilon+\delta}{(1+\delta)^2} \right\rangle$ (in general, it will intersect R_{23}, but we do not need to prove this). Let $\boldsymbol{v}_1 := \langle \epsilon + \delta, 1 - \epsilon, -(1+\delta) \rangle$, a (non-unit) vector that defines the plane of A_1. We may define $A_2, A_3, \boldsymbol{v}_2, \boldsymbol{v}_3$ similarly.

Lemma 7.2.4. *For every $\boldsymbol{x} \in A_1$, we have $(d\boldsymbol{x}/dt) \cdot \boldsymbol{v}_1 \geq 0$.*

Proof of Lemma. The surface A_1 has possibly nonempty intersection with $R'_\emptyset, R'_{23}, R'_{123}$. We consider these cases separately. For $\boldsymbol{x} \in R'_\emptyset$, the result follows from Lemma 7.2.3. For $\boldsymbol{x} \in R'_{23}$, we have:

$$\begin{aligned} \frac{d\boldsymbol{x}}{dt} \cdot \boldsymbol{v}_1 &= -\boldsymbol{x} \cdot \boldsymbol{v}_1 + (\epsilon + \delta)(W_{1\bullet}\boldsymbol{x} + \theta) \\ &= 0 + (\epsilon + \delta)(W_{1\bullet}\boldsymbol{x} + \theta) \\ &\geq 0. \end{aligned}$$

Finally, for $\boldsymbol{x} \in R'_{123}$, we have $(d\boldsymbol{x}/dt) \cdot \boldsymbol{v}_1 = -\boldsymbol{x} \cdot \boldsymbol{v}_1 = 0$. Thus, the Lemma holds in all cases. □

Proof of Theorem 7.2.1. Define A as the identification of A_1, A_2, A_3 under the natural quotient map π. We define the function $f : A \to A$ such that if $\boldsymbol{x}(0) \in A_i$, then $f(\pi(\boldsymbol{x}(0))) = \pi(\boldsymbol{x}(t))$, where t is the minimal value such that $\boldsymbol{x}(t) \in A_{i+1}$ (where subscripts are taken modulo 3). It follows from Lemma 7.2.4 that the function f is well-defined and continuous, where we also use the fact that there is no fixed point for $\boldsymbol{x}(t)$ apart from $\boldsymbol{y}$. Therefore, by the Brouwer fixed point theorem, there exists a fixed point of the function f, which corresponds to a periodic trajectory for $\boldsymbol{x}(t)$. □

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Noah Apthorpe, Alexander Riordan, Robert Aguilar, Jan Homann, Yi Gu, David Tank, and H Sebastian Seung. Automatic neuron detection in calcium imaging data using convolutional networks. In *Advances In Neural Information Processing Systems (NIPS)*, pages 3270–3278, 2016.

[3] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1120–1128, 2016.

[4] 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.

[5] Baktash Babadi and Haim Sompolinsky. Sparseness and expansion in sensory representations. *Neuron*, 83(5):1213–1226, 2014.

[6] Omri Barak, Mattia Rigotti, and Stefano Fusi. The sparseness of mixed selectivity neurons controls the generalization–discrimination trade-off. *The Journal of Neuroscience*, 33(9):3844–3856, 2013.

[7] Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14(1):115–133, 1994.

[8] Christopher J Bates, Ilker Yildirim, Joshua B Tenenbaum, and Peter W Battaglia. Humans predict liquid dynamics using probabilistic simulation. In *Proceedings of the Conference of the Cognitive Science Society*, 2015.

[9] Jeffrey M Beck, Wei Ji Ma, Xaq Pitkow, Peter E Latham, and Alexandre Pouget. Not noisy, just wrong: the role of suboptimal inference in behavioral variability. *Neuron*, 74(1):30–39, 2012.

[10] Eyal Beigman and Beata Beigman Klebanov. Learning with annotation noise. In *Proceedings of the Joint Conference of the Annual Meeting of the ACL and the International Joint Conference on Natural Language Processing of the AFNLP*. Association for Computational Linguistics, 2009.

[11] R Ben-Yishai, R Lev Bar-Or, and H Sompolinsky. Theory of orientation tuning in visual cortex. *Proceedings of the National Academy of Sciences*, 92(9):3844–3848, 1995.

[12] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *Preprint arXiv:1502.04156*, 2015.

[13] Ari S Benjamin, David Rolnick, and Konrad Kording. On the behavior and regularization of networks in function space. *Submitted*, 2018.

[14] Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems*, 25(8):1553–1565, 2014.

[15] Patrick Billingsley. *Probability and measure*. John Wiley & Sons, 2008.

[16] Elizabeth Bonawitz, Stephanie Denison, Alison Gopnik, and Thomas L. Griffiths. Win-Stay, Lose-Sample: A simple sequential algorithm for approximating Bayesian inference. *Cognitive Psychology*, 74:35–65, 2014.

[17] Carla E Brodley and Mark A Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research (JAIR)*, 11:131–167, 1999.

[18] David Budden, Alexander Matveev, Shibani Santurkar, Shraman Ray Chaudhuri, and Nir Shavit. Deep tensor convolution on multicores. *arXiv preprint arXiv:1611.06565*, 2016.

[19] Fei Chen, Paul W Tillberg, and Edward S Boyden. Expansion microscopy. *Science*, 347(6221):543–548, 2015.

[20] Jianxu Chen, Lin Yang, Yizhe Zhang, Mark Alber, and Danny Z Chen. Combining fully convolutional and recurrent neural networks for 3d biomedical image segmentation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3036–3044, 2016.

[21] François Chollet et al. Keras. `https://github.com/keras-team/keras`, 2018.

[22] Charles K Chui, Xin Li, and Hrushikesh Narhar Mhaskar. Limitations of the approximation capabilities of neural networks with one hidden layer. *Advances in Computational Mathematics*, 5(1):233–243, 1996.

[23] CK Chui, Xin Li, and HN Mhaskar. Neural networks for localized approximation. *Mathematics of Computation*, 63(208):607–623, 1994.

[24] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. *Journal of Machine Learning Research (JMLR)*, 49, 2016.

[25] Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 955–963, 2016.

[26] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, (3):326–334, 1965.

[27] Carina Curto, Jesse Geneson, and Katherine Morrison. Fixed points of threshold-linear networks, 2018.

[28] Carina Curto and Katherine Morrison. Pattern completion in symmetric threshold-linear networks. *Neural Computation*, 28(12):2825–2852, 2016.

[29] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[30] Amit Daniely. Depth separation for neural networks. In *Conference On Learning Theory (COLT)*, 2017.

[31] Jesús A De Loera, Reuben N La Haye, David Rolnick, and Pablo Soberón. Quantitative combinatorial geometry for continuous parameters. *Discrete & Computational Geometry*, 57(2):318–334, 2017.

[32] Jesús A De Loera, Reuben N La Haye, David Rolnick, and Pablo Soberón. Quantitative Tverberg theorems over lattices and other discrete sets. *Discrete & Computational Geometry*, 58(2):435–448, 2017.

[33] Jesús A De Loera, Susan Margulies, Michael Pernpeintner, Eric Riedl, David Rolnick, Gwen Spencer, Despina Stasi, and Jon Swenson. Graph-coloring ideals: Nullstellensatz certificates, Gröbner bases for chordal graphs, and hardness of Gröbner bases. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 133–140. ACM, 2015.

[34] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 666–674, 2011.

[35] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[36] Stephanie Denison, Elizabeth Bonawitz, Alison Gopnik, and Thomas L. Griffiths. Rational variability in children's causal inferences: The Sampling Hypothesis. *Cognition*, 126(2):280–300, 2013.

[37] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Annual Conference on Learning Theory (COLT)*, pages 907–940, 2016.

[38] Roozbeh Farhoodi, Pavan Ramkumar, and Konrad Kording. Deep learning approach towards generating neuronal morphology. Computational and Systems Neuroscience (Cosyne), 2017.

[39] Roozbeh Farhoodi, David Rolnick, and Konrad Kording. Neuron dendrograms reflect an asymmetrical growth process. *Computational and Systems Neuroscience (Cosyne)*, 2018.

[40] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.

[41] Rong Ge, Chi Jin, and Yi Zheng. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. *arXiv preprint arXiv:1704.00708*, 2017.

[42] Samuel J Gershman, Edward Vul, and Joshua B Tenenbaum. Multistability and perceptual inference. *Neural computation*, 24(1):1–24, 2012.

[43] Raja Giryes, Guillermo Sapiro, and Alexander M Bronstein. Deep neural networks with random Gaussian weights: a universal classification strategy? *IEEE Trans. Signal Processing*, 64(13):3444–3457, 2016.

[44] Noah Golowich and David Rolnick. Acyclic subgraphs of planar digraphs. *The Electronic Journal of Combinatorics*, 22(3):P3–7, 2015.

[45] Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in Cognitive Sciences*, 14(8):357–364, 2010.

[46] Melody Y Guan, Varun Gulshan, Andrew M Dai, and Geoffrey E Hinton. Who said what: Modeling individual labelers improves classification. *arXiv:1703.08774*, 2017.

[47] Stephen Handel. *Listening: An introduction to the perception of auditory events.* The MIT Press, 1993.

[48] Boris Hanin. Universal function approximation by deep neural nets with bounded width and ReLU activations. *arXiv preprint arXiv:1708.02691*, 2017.

[49] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? *arXiv preprint arXiv:1801.03744*, 2018.

[50] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. *arXiv preprint arXiv:1803.01719*, 2018.

[51] Boris Hanin and Mark Sellke. Approximating continuous functions by ReLU nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[53] 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 (CVPR)*, pages 770–778, 2016.

[54] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[55] Moritz Helmstaedter, Kevin L Briggman, Srinivas C Turaga, Viren Jain, H Sebastian Seung, and Winfried Denk. Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500(7461):168–174, 2013.

[56] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 48, pages 2034–2042, 2016.

[57] David Grant Colburn Hildebrand, Marcelo Cicconet, Russel Miguel Torres, Woohyuk Choi, Tran Minh Quan, Jungmin Moon, Arthur Willis Wetzel, Andrew Scott Champion, Brett Jesse Graham, Owen Randlett, et al. Whole-brain serial-section electron microscopy in larval zebrafish. *Nature*, 2017.

[58] Christopher Hillar, Jascha Sohl-Dickstein, and Kilian Koepsell. Efficient and optimal binary Hopfield associative memory storage using minimum probability flow. *arXiv preprint arXiv:1204.2916*, 2012.

[59] 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.

[60] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[61] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[62] Viren Jain, Srinivas C Turaga, K Briggman, Moritz N Helmstaedter, Winfried Denk, and H Sebastian Seung. Learning to agglomerate superpixel hierarchies. In *Advances in Neural Information Processing Systems (NIPS)*, pages 648–656, 2011.

[63] Michał Januszewski, Jeremy Maitin-Shepard, Peter Li, Jörgen Kornfeld, Winfried Denk, and Viren Jain. Flood-filling networks. *arXiv preprint arXiv:1611.00421*, 2016.

[64] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M Kakade, and Michael I Jordan. How to escape saddle points efficiently. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1724–1732, 2017.

[65] Dezhe Z. Jin and Alexay A. Kozhevnikov. A compact statistical model of the song syntax in Bengalese finch. *PLoS Comput Biol*, 7(3):1–19, 03 2011.

[66] Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (EUNN) and their application to RNNs. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1733–1741, 2017.

[67] Jeffrey S Johnson and Bruno A Olshausen. The recognition of partially visible natural objects in the presence and absence of their occluders. *Vision research*, 45(25):3262–3276, 2005.

[68] Armand Joulin, Laurens van der Maaten, Allan Jabri, and Nicolas Vasilache. Learning visual features from large weakly supervised data. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016.

[69] Jonathan Kadmon and Haim Sompolinsky. Transition to chaos in random neuronal networks. *Physical Review X*, 5(4):041030, 2015.

[70] Narayanan Kasthuri, Kenneth Jeffrey Hayworth, Daniel Raimund Berger, Richard Lee Schalek, José Angel Conchello, Seymour Knowles-Barley, Dongil Lee, Amelio Vázquez-Reina, Verena Kaynig, Thouis Raymond Jones, et al. Saturated reconstruction of a volume of neocortex. *Cell*, 162(3):648–661, 2015.

[71] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems (NIPS)*, pages 586–594, 2016.

[72] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[73] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[74] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 972–981, 2017.

[75] Graham Knott, Herschel Marchman, David Wall, and Ben Lich. Serial section scanning electron microscopy of adult brain tissue using focused ion beam milling. *Journal of Neuroscience*, 28(12):2959–2964, 2008.

[76] I Krasin, T Duerig, N Alldrin, A Veit, S Abu-El-Haija, S Belongie, D Cai, Z Feng, V Ferrari, V Gomes, et al. Openimages: A public dataset for large-scale multi-label and multiclass image classification. *Dataset available from https://github. com/openimages*, 2(6):7, 2016.

[77] Jonathan Krause, Benjamin Sapp, Andrew Howard, Howard Zhou, Alexander Toshev, Tom Duerig, James Philbin, and Li Fei-Fei. The unreasonable effectiveness of noisy data for fine-grained recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016.

[78] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[79] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[80] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013.

[81] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[82] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The MNIST database of handwritten digits, 1998.

[83] Kisuk Lee, Aleksandar Zlateski, Vishwanathan Ashwin, and H Sebastian Seung. Recursive training of 2d-3d convolutional networks for neuronal boundary prediction. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3573–3581, 2015.

[84] Jeff W Lichtman, Hanspeter Pfister, and Nir Shavit. The big data challenges of connectomics. *Nature Neuroscience*, 17(11):1448–1454, 2014.

[85] 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.

[86] Michael A. Long and Michale S. Fee. Using temperature to analyse temporal dynamics in the songbird motor pathway. *Nature*, 456(7219):189–194, 11 2008.

[87] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6232–6240, 2017.

[88] Wei Ji Ma, Jeffrey M Beck, and Alexandre Pouget. Spiking networks for Bayesian inference and choice. *Current Opinion in Neurobiology*, 18(2):217–222, 2008.

[89] Jeremy B Maitin-Shepard, Viren Jain, Michal Januszewski, Peter Li, and Pieter Abbeel. Combinatorial energy learning for image segmentation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1966–1974, 2016.

[90] Naresh Manwani and PS Sastry. Noise tolerance under risk minimization. *IEEE Transactions on Cybernetics*, 43(3):1146–1151, 2013.

[91] Adam H Marblestone, Greg Wayne, and Konrad P Kording. Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience*, 10, 2016.

[92] James Martens, Arkadev Chattopadhya, Toni Pitassi, and Richard Zemel. On the representational efficiency of restricted Boltzmann machines. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2877–2885, 2013.

[93] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[94] Alexander Matveev, Yaron Meirovitch, Hayk Saribekyan, Wiktor Jakubiuk, Tim Kaler, Gergely Odor, David Budden, Aleksandar Zlateski, and Nir Shavit. A multicore path to connectomics-on-demand. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 267–281. ACM, 2017.

[95] Yaron Meirovitch, Alexander Matveev, Hayk Saribekyan, David Budden, David Rolnick, Gergely Odor, Seymour Knowles-Barley Jones, Raymond Thouis, Hanspeter Pfister, Jeff William Lichtman, and Nir Shavit. A multi-pass approach to large-scale connectomics. *arXiv preprint arXiv:1612.02120*, 2016.

[96] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. Learning functions: When is deep better than shallow. *arXiv:1603.00988v4*, 2016.

[97] Hrushikesh Narhar Mhaskar. Approximation properties of a multilayered feedforward artificial neural network. *Advances in Computational Mathematics*, 1(1):61–80, 1993.

[98] Ishan Misra, C Lawrence Zitnick, Margaret Mitchell, and Ross Girshick. Seeing through the human reporting bias: Visual classifiers from noisy human-centric labels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2930–2939, 2016.

[99] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2924–2932, 2014.

[100] Josh Lyskowski Morgan, Daniel Raimund Berger, Arthur Willis Wetzel, and Jeff William Lichtman. The fuzzy logic of network connectivity in mouse visual thalamus. *Cell*, 165(1):192–206, 2016.

[101] Katherine Morrison, Anda Degeratu, Vladimir Itskov, and Carina Curto. Diversity of emergent dynamics in competitive threshold-linear networks: a preliminary report. *arXiv preprint arXiv:1605.04463*, 2016.

[102] Richard A Moy and David Rolnick. Novel structures in Stanley sequences. *Discrete Mathematics*, 339(2):689–698, 2016.

[103] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1196–1204, 2013.

[104] Radford M Neal. Probabilistic inference using Markov chain Monte Carlo methods. 1993.

[105] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2014–2023, 2016.

[106] Bence P Ölveczky, Aaron S Andalman, and Michale S Fee. Vocal experimentation in the juvenile songbird requires a basal ganglia circuit. *PLoS Biol*, 3(5):e153, 2005.

[107] Toufiq Parag, Anirban Chakraborty, Stephen Plaza, and Louis Scheffer. A context-aware delayed agglomeration framework for electron microscopy segmentation. *PloS one*, 10(5):e0125825, 2015.

[108] Toufiq Parag, Dan C Ciresan, and Alessandro Giusti. Efficient classifier training to minimize false merges in electron microscopy segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 657–665, 2015.

[109] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[110] Dejan Pecevski, Lars Buesing, and Wolfgang Maass. Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons. *PLoS Comput Biol*, 7(12):e1002294, 2011.

[111] Ian D Peikon, Justus M Kebschull, Vasily V Vagin, Diana I Ravens, Yu-Chi Sun, Eric Brouzes, Ivan R Corrêa, Dario Bressan, and Anthony Zador. Using high-throughput barcode sequencing to efficiently map connectomes. *bioRxiv*, page 099093, 2017.

[112] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4788–4798, 2017.

[113] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[114] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016.

[115] Stephen M Plaza and Stuart E Berg. Large-scale electron microscopy image segmentation in Spark. *arXiv preprint arXiv:1604.00385*, 2016.

[116] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, 15(5):503–519, 2017.

[117] 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 (NIPS)*, pages 3360–3368, 2016.

[118] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. Survey of expressivity in deep neural networks. *arXiv:1611.08083*, 2016.

[119] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2847–2854, 2017.

[120] David Rolnick. The on-line degree ramsey number of cycles. *Discrete Mathematics*, 313(20):2084–2093, 2013.

[121] David Rolnick. On the classification of Stanley sequences. *European Journal of Combinatorics*, 59:51–70, 2017.

[122] David Rolnick, Kevin Aydin, Shahab Kamali, Vahab Mirrokni, and Amir Najmi. GeoCUTS: Geographic Clustering Using Travel Statistics. *arXiv:1611.03780*, 2016.

[123] David Rolnick, Yaron Meirovitch, Toufiq Parag, Hanspeter Pfister, Viren Jain, Jeff W Lichtman, Edward S Boyden, and Nir Shavit. Morphological error detection in 3D segmentations. *arXiv:1705.10882*, 2017.

[124] David Rolnick and Pablo Soberón. Algorithmic aspects of Tverberg's theorem. *arXiv:1601.03083*, 2016.

[125] David Rolnick and Pablo Soberón. Quantitative (p, q) theorems in combinatorial geometry. *Discrete Mathematics*, 340(10):2516–2527, 2017.

[126] David Rolnick and Max Tegmark. The power of deeper networks for expressing natural functions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[127] David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv:1705.10694*, 2017.

[128] David Rolnick and Praveen S Venkataramana. On the growth of Stanley sequences. *Discrete Mathematics*, 338(11):1928–1937, 2015.

[129] William Gray Roncal, Michael Pekala, Verena Kaynig-Fittkau, Dean M Kleissas, Joshua T Vogelstein, Hanspeter Pfister, Randal Burns, R Jacob Vogelstein, Mark A Chevillet, and Gregory D Hager. Vesicle: Volumetric evaluation of synaptic interfaces using computer vision at large scale. *arXiv preprint arXiv:1403.3724*, 2014.

[130] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[131] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[132] Adam N Sanborn and Thomas L Griffiths. A Bayesian Framework for Modeling Intuitive Dynamics. In *Proceedings of the Conference of the Cognitive Science Society*, pages 1145–1150, 2009.

[133] Shibani Santurkar, David Budden, Alexander Matveev, Heather Berlin, Hayk Saribekyan, Yaron Meirovitch, and Nir Shavit. Toward streaming synapse detection with compositional convnets. *arXiv preprint arXiv:1702.07386*, 2017.

[134] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[135] Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. 2017.

[136] Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. A correspondence between random neural networks and statistical field theory. *arXiv preprint arXiv:1710.06570*, 2017.

[137] H. Sebastian Seung, Daniel D. Lee, Ben Y. Reis, and David W. Tank. The autapse: A simple illustration of short-term analog memory storage by tuned synaptic feedback. *Journal of Computational Neuroscience*, 9(2):171–185, 2000.

[138] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3067–3075, 2017.

[139] Ramalingam Shanmugam and Rajan Chattamvelli. *Statistics for scientists and engineers.* Wiley Online Library, 2015.

[140] Lei Shi, Thomas L Griffiths, Naomi H Feldman, and Adam N Sanborn. Exemplar models as a mechanism for performing Bayesian inference. *Psychonomic Bulletin & Review*, 17(4):443–464, 2010.

[141] Haim Sompolinsky and I Kanter. Temporal association in asymmetric neural networks. *Physical Review Letters*, 57(22):2861, 1986.

[142] Gwen Spencer and David Rolnick. On the robust hardness of Gröbner basis computation. *arXiv:1511.06436*, 2015.

[143] Amos Storkey. Increasing the capacity of a Hopfield network without sacrificing functionality. In *International Conference on Artificial Neural Networks*, pages 451–456. Springer, 1997.

[144] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 945–953, 2015.

[145] Sainbayar Sukhbaatar, Joan Bruna, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. Training convolutional networks with noisy labels. *arXiv preprint arXiv:1406.2080*, 2014.

[146] Uygar Sümbül, Douglas Roossien, Dawen Cai, Fei Chen, Nicholas Barry, John P Cunningham, Edward Boyden, and Liam Paninski. Automated scalable segmentation of neurons from multispectral images. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1912–1920, 2016.

[147] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 843–852. IEEE, 2017.

[148] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[149] Shin-ya Takemura, Arjun Bharioke, Zhiyuan Lu, Aljoscha Nern, Shiv Vitaladevuni, Patricia K Rivlin, William T Katz, Donald J Olbris, Stephen M Plaza, Philip Winston, et al. A visual motion detection circuit suggested by Drosophila connectomics. *Nature*, 500(7461):175–181, 2013.

[150] Masato Taki. Deep residual networks and weight initialization. *arXiv preprint arXiv:1709.02956*, 2017.

[151] Matus Telgarsky. Benefits of depth in neural networks. *Journal of Machine Learning Research (JMLR)*, 49, 2016.

[152] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. YFCC100M: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[153] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 595–604, 2015.

[154] Andreas Veit, Neil Alldrin, Gal Chechik, Ivan Krasin, Abhinav Gupta, and Serge Belongie. Learning from noisy large-scale datasets with minimal supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6575–6583, 2017.

[155] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 550–558, 2016.

[156] Ed Vul, George Alvarez, Joshua B Tenenbaum, and Michael J Black. Explaining human multiple object tracking as resource-constrained approximate inference in a dynamic probabilistic model. In *Advances in Neural Information Processing Systems*, pages 1955–1963, 2009.

[157] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2794–2802, 2015.

[158] Yuhuai Wu, Elman Mansimov, Roger B. Grosse, Shun Liao, and Jimmy Ba. Second-order optimization for deep reinforcement learning using Kronecker-factored approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5285–5294, 2017.

[159] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[160] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. Learning from massive noisy labeled data for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2691–2699, 2015.

[161] Rafael Yuste, Jason N MacLean, Jeffrey Smith, and Anders Lansner. The cortex as a central pattern generator. *Nature Reviews Neuroscience*, 6(6):477, 2005.

[162] Matthew D Zeiler. ADADELTA: an adaptive learning rate method. *arXiv:1212.5701*, 2012.

[163] 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)*, 2017.

[164] Ting Zhao and Stephen M Plaza. Automatic neuron type identification by neurite localization in the Drosophila medulla. *arXiv preprint arXiv:1409.1892*, 2014.

[165] Xiaojin Zhu. Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison*, 2(3):4, 2005.

[166] Aleksandar Zlateski, Kisuk Lee, and H Sebastian Seung. Scalable training of 3d convolutional networks on multi- and many-cores. *Journal of Parallel and Distributed Computing*, 2017.