Analysis of Gradient Based Learning in Deep Networks: A Case Study

A Project Report

submitted by

RAJAT VADIRAJ DWARAKNATH

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2020

THESIS CERTIFICATE

This is to certify that the thesis titled **Analysis of Gradient Based Learning in Deep Networks: A Case Study**, submitted by **Rajat Vadiraj Dwaraknath (EE16B033)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Harish Guruprasad
Research Guide
Assistant Professor
Dept. of Computer Science
IIT-Madras, 600 036

Prof. Abhishek Sinha
Department Co-Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

ACKNOWLEDGEMENTS

I would like to deeply express my gratitude towards Prof. Harish Guruprasad, my mentor and guide without whom this project would not have been possible. The regular meetings combined with his invaluable advice and guidance not only helped me in this endeavour but also molded my outlook towards research as a whole. I would additionally like to thank the other graduate and undergraduate students that are part of Prof. Harish's research group for the numerous thoughtful and enlightening discussions we have had. I would also like to thank Prof. Abhishek Sinha for agreeing to be my department co-guide for this project. Last but not least, I would like to thank my friends and family for their unending encouragement and continued support as I worked on this project.

ABSTRACT

KEYWORDS: deep learning theory, neural network, optimization, generalization

We analyze the dynamics of gradient descent for training the simplest three-layer neural network architecture for which no formal theoretical explanation has been formulated yet. We motivate this architecture by building on existing two-layer network theory and combining it with ideas based on deep linear networks. We formulate numerous testable hypotheses and present empirical results for them. The setting of the case study affords the ability to create metrics to measure intermediate layer performance that act as probes to better understand the learning dynamics of the network. We present additional testable hypotheses based on this idea that would normally not be testable in common deep learning settings, and the results of these experiments reveal promising directions to pursue for future theoretical work.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

NTK	Neural Tangent Kernel
PL condition	Polyak-Lojasiewicz condition
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
CNN	Convolutional Neural Network

CHAPTER 1

Introduction

Deep learning has seen remarkable advancements in its practical utility in a wide variety of fields encompassing image processing, natural language processing, medical diagnostics, etc. Although, much of this progress could be described as alchemical and the theoretical foundations for empirical observations lag behind greatly. However, recent research has cast some light on the inner workings of very specific and simple classes of neural network architectures. Specifically, significant theory has been formulated for the analysis of two-layer neural networks. While some form of extensions can be made to deeper architectures, they are less convincing. Deep linear networks have also been a focus of study and have shown fruitful theoretical developments. In this project, we aim to tackle the next most simple type of architecture by combining these ideas. We study the specific case of a three-layer network in which the first layer has a linear activation and a bottleneck width of 1. This represents the simplest possible three-layer architecture with non-trivial learning dynamics under gradient descent that does not have a formal theoretical analysis yet. We test numerous hypotheses in hopes of shedding light on possible directions to pursue to develop a theory for this class of models. An additional advantage of this case study is the ability to probe intermediate stages of the network and create metrics to measure performance in ways that would normally not be possible in other common deep learning settings like training a CNN on MNIST. This allows us to test additional hypotheses related to parameter control and enables a finer study of phases in training process.

Chapter 2 presents a brief summary of important previous work upon which some of the hypotheses build on in later sections. Chapter 3 describes in detail the setting of the specific case study. Finally, Chapter 4 presents various hypotheses in the context of the case study, from comparing to basic 1-D learning baselines to analyzing the effect of controlling different parts of the network. Methods to test these hypotheses and their results are also presented and analyzed.

CHAPTER 2

Background

Deep learning has made remarkable progress from a practical standpoint in the recent past. However, the theoretical side of this field has been lagging behind the empirical advancements significantly. [26] pointed out that the primary aspect in existing theories which make them fail when used in deep learning settings is their inability to explain the remarkable generalization performance of deep networks even when overparameterized. Overparameterization is seen to be a blessing rather than a curse as most practical neural networks used in image processing [24], natural language processing [25], etc operate in the overparameterized regime. This unexpected behavior has eluded theoretical explanations, although recent work has made inroads into understanding this phenomenon.

2.1 The Neural Tangent Kernel

A flurry of recent papers [14, 13, 2, 19, 1, 27, 17] study the behavior of neural networks in the *over-parameterized, infinite width* limit, and prove convergence guarantees for gradient descent in this limit. A common flavour of mathematical tools is used to arrive at these results – neural networks approach their linear approximation around their initializations in this limit and behave as kernel machines (this regime is termed *lazy training*). The paper [10] captures a more basic criterion for this approximation to hold true, which generalizes to arbitrary non-linear models and is not restricted to neural networks of specific architectures as in the other papers. We present a brief summary of this analysis in this section.

2.1.1 Setup

- **Model:** Consider a simple twice-differentiable non-linear model (could be any twice-differentiable neural network) $f : \mathbb{R}^p \times \mathbb{R}^d \to \mathbb{R}$, with p parameters and an output dimension of 1.

- **Data:** Denote the dataset of size n by $\{\tilde{\mathbf{x}}_\mathbf{i}, \tilde{y}_i\}_{i=1}^n$. Arrange the labels $\tilde{y}_i$ into a single vector $\tilde{\mathbf{y}} \in \mathbb{R}^n$. For a fixed set of parameters $\mathbf{w} \in \mathbb{R}^p$, let the vector of model outputs $\mathbf{y}(\mathbf{w}) \in \mathbb{R}^n$ be given by $y_i(\mathbf{w}) = f(\mathbf{x_i}, \mathbf{w})$. We can also deal with the case where $\mathbf{y}(\mathbf{w})$ is an element in an infinite dimensional function space (Hilbert space), but this complicates the analysis. Results on the finite dimensional case carry over to the general case without many issues.
- **Loss:** We restrict our study to full-batch gradient descent on the square loss, so our loss is simply:

$$L(\mathbf{w}) = \frac{1}{2}\|\mathbf{y}(\mathbf{w}) - \tilde{\mathbf{y}}\|_2^2$$

Experimental results indicate that the weights of very wide neural networks barely change during training. This motivates us to look at the *linearization* of the model with respect to the *parameters* $\mathbf{w}$ around the initialization $\mathbf{w_0}$:

$$f(x, \mathbf{w}) \approx f(x, \mathbf{w_0}) + \nabla_\mathbf{w} f(x, \mathbf{w_0})^T (\mathbf{w} - \mathbf{w_0})$$

In the vector notation introduced above:

$$\mathbf{y}(\mathbf{w}) \approx y(\mathbf{w_0}) + \nabla_\mathbf{w} \mathbf{y}(\mathbf{w_0})^T (\mathbf{w} - \mathbf{w_0})$$

Note that the Jacobian $\nabla_\mathbf{w} \mathbf{y}(\mathbf{w})$ has size $p \times n$. The important point to notice is that this model is linear in the parameters $\mathbf{w}$, so minimizing the least squares loss reduces to linear regression. This linearized model is a kernel machine where the corresponding **feature map** $\phi(x)$ is obtained by taking the gradient of the model outputs at initialization with respect to its weights.

$$\phi(x) = \nabla_\mathbf{w} f(x, \mathbf{w_0})$$

2.1.2 General Criterion

A general criterion for when the linear approximation is accurate derived in [10]. Here we present a rough intuitive derivation. As we train the model (with small enough learning rate), we can safely assume that the model outputs $\mathbf{y}(\mathbf{w})$ always get closer to the target outputs $\tilde{\mathbf{y}}$. So we can obtain the following bound (all norms are ℓ_2 or operator norms):

$$\text{net change in } \mathbf{y}(\mathbf{w}) \lesssim \|\mathbf{y}(\mathbf{w_0}) - \tilde{\mathbf{y}}\|$$

Now we can quantify the *distance* we move in parameter space with a first order approximation,

$$\text{distance } d \text{ moved in } w \text{ space} \approx \frac{\text{net change in } \mathbf{y}(\mathbf{w})}{\text{rate of change of } \mathbf{y} \text{ w.r.t } \mathbf{w}} = \frac{\|\mathbf{y}(\mathbf{w_0}) - \tilde{\mathbf{y}}\|}{\|\nabla \mathbf{y}(\mathbf{w_0})\|}$$

Translating to change in the Jacobian, we get:

$$\text{change in model Jacobian} \approx \text{distance } d \times \text{rate of change of the Jacobian} = d \cdot \|\nabla^2 \mathbf{y}(\mathbf{w_0})\|$$

Where $\nabla^2 \mathbf{y}(\mathbf{w_0})$ is a rank-3 Hessian tensor. The model is linear when the *relative change* in its Jacobian is very small.

$$\begin{aligned}\text{relative change in Jacobian} &\approx \frac{d \cdot \text{rate of change of Jacobian}}{\text{norm of Jacobian}} \\ &= \frac{d \cdot \|\nabla^2 \mathbf{y}(\mathbf{w_0})\|}{\|\nabla \mathbf{y}(\mathbf{w_0})\|} \\ &= \|(\mathbf{y}(\mathbf{w}_0) - \tilde{\mathbf{y}})\| \frac{\|\nabla^2 \mathbf{y}(\mathbf{w_0})\|}{\|\nabla \mathbf{y}(\mathbf{w_0})\|^2} \ll 1\end{aligned}$$

The quantity in blue is called the *inverse relative scale* $\kappa(\mathbf{w_0})$ of the model. Observe that this quantity can be driven arbitrarily close to 0 by simply **scaling the model outputs** by a factor α, assuming that the model outputs are zero at initialization. ($\mathbf{y}(\mathbf{w_0}) = 0$). This can always be achieved by simply defining a new model which subtracts a fixed value equal to the initial outputs from the original model.) The rescaled model is $\mathbf{y}'(\mathbf{w_0}) = \alpha \mathbf{y}(\mathbf{w_0})$.

$$\kappa_\alpha(\mathbf{w_0}) = \frac{\|\nabla^2 \alpha \mathbf{y}(\mathbf{w_0})\|}{\|\nabla \alpha \mathbf{y}(\mathbf{w_0})\|^2} = \frac{1}{\alpha} \frac{\|\nabla^2 \mathbf{y}(\mathbf{w_0})\|}{\|\nabla \mathbf{y}(\mathbf{w_0})\|^2} \to 0 \text{ as } \alpha \to \infty$$

Note that such rescaling can also be obtained by scaling the initialization in the case of homogeneous models. For the specific case of *1-hidden layer* neural networks of with hidden width m initialized with Gaussian weights $\mathcal{N}(0, \frac{1}{m})$ (LeCun initialization), we can show that $\kappa_m(\mathbf{w_0}) \sim \mathcal{O}(\frac{1}{m})$, which means that the neural network approaches the linearized regime as the width goes to infinity. The proof of this involves messing around with Hessians of the neural network and we refer the reader to [15] for the detailed derivation.

2.1.3 Gradient Flow

We can approximate gradient descent by taking the learning rate to be infinitesimally small and replacing it with a differential equation. This is called the gradient flow of the parameters (dot denotes time derivative, ∇ denotes gradient w.r.t weights $\mathbf{w}$):

$$\dot{\mathbf{w}}(t) = -\nabla L(\mathbf{w}(t))$$

We will drop the time variable as it can be inferred from context. Substituting for the loss, we get:

$$\dot{\mathbf{w}} = -\nabla \mathbf{y}(\mathbf{w})(\mathbf{y}(\mathbf{w}) - \tilde{\mathbf{y}})$$

The dynamics of the model outputs $\mathbf{y}(\mathbf{w})$ induced by this gradient flow can be derived using the chain rule:

$$\dot{\mathbf{y}}(\mathbf{w}) = \nabla \mathbf{y}(\mathbf{w})^T \dot{\mathbf{w}} = -\nabla \mathbf{y}(\mathbf{w})^T \nabla \mathbf{y}(\mathbf{w})(\mathbf{y}(\mathbf{w}) - \tilde{\mathbf{y}})$$

The quantity $\mathbf{H}(\mathbf{w}) := \nabla \mathbf{y}(\mathbf{w})^T \nabla \mathbf{y}(\mathbf{w})$ is called the *neural tangent kernel* (NTK). In the lazy regime ($\kappa(\mathbf{w_0}) \ll 1$), the Jacobian of the model outputs does not change as training progresses. In other words,

$$\nabla \mathbf{y}(\mathbf{w}(t)) \approx \nabla \mathbf{y}(\mathbf{w}_0) \implies \mathbf{H}(\mathbf{w}(t)) \approx \mathbf{H}(\mathbf{w}_0)$$

The training dynamics now reduces to a very simple linear ordinary differential equation:

$$\dot{\mathbf{y}}(\mathbf{w}) = -\mathbf{H}(\mathbf{w}_0)(\mathbf{y}(\mathbf{w}) - \tilde{\mathbf{y}})$$

The analysis of this linear ODE is straightforward by performing a spectral decomposition of the positive (semi) definite NTK. There are two cases:

- **Under-parameterized:** $p < n$ which means that $\mathbf{H}(\mathbf{w}_0)$ is not full rank and has some 0 eigenvalues. This means that the above dynamics converges to a local minimum.
- **Over-parameterized:** $p > n$ which, assuming non-degeneracy of the dataset, means that $\mathbf{H}(\mathbf{w}_0)$ is full rank and positive definite. All eigenvalues are positive. Clearly, this implies that $\mathbf{y}(\mathbf{w}(\infty)) = \tilde{\mathbf{y}}$, and the model always converges to 0 training loss.

We have shown that the linearized model achieves 0 training loss in the overparame-

terized case. By bounding the deviation of the non-linear trajectories from that of the linearized model, we can show that the non-linear model also converges to 0 training loss. Results with quantitative bounds for a finite trajectory duration in terms of the model scale α can be found in [10]. The take home point is that these deviations go to 0 as $\alpha \to \infty$, allowing one to carry the convergence results from the linearized case to the non-linear case. Thus, we have shown that over-parameterized models converge to 0 training loss in the lazy regime.

2.1.4 Two-Layer Network Results

We have already mentioned that one can derive quantitatively the relationship between the width of a two-layer network (or one-hidden layer) and its inverse relative scale (see [15]). In addition to this, more detailed research has been conducted in analyzing two-layer networks from the perspective of NTK theory. In particular, [6] develops a tighter characterization of the speed of training and provides an explanation for the observations related to slower training with random labels presented in [26]. While the NTK theory described so far has been purely optimization related, [6] also develops data-dependent generalization bounds based on the alignment between the labels and the top eigenvectors of the inverse NTK. This generalization bound differs from other common bounds in that it has no dependence on the width of the network. They also prove that a certain class of smooth functions are learnable by two-layer networks. These results present a strong baseline on which to explore further developments.

2.2 Results outside the NTK regime

While the NTK theory affords simple and accurate theoretical results for optimization, it fails to appear in practical experiments. Most neural network architectures which broke state of the art barriers consistently operate *outside* the NTK regime, and their tangent kernels significantly change during the course of gradient descent. Additionally, these practical networks greatly outperform the kernel machines which use the corresponding NTK [5]. These empirical results indicate that something more than the NTK theory is needed to fully explain the properties of practical neural networks. Although no convincing theory for practically used large scale networks has arisen, some progress

has been made in this regard for very specific architectures.

2.2.1 Global Convergence using Optimal Transport

In [9], the authors study a specific two-layer neural network architecture and obtain global convergence guarantees for gradient descent with some regularity assumptions. However, their result holds only in the infinite width limit, and it is not quantitative – both in terms of width and convergence rate. However, the training dynamics in this regime is shown to not be in the NTK regime. We present a brief summary of these ideas in this section, based on the accompanying blog post by the authors [7].

Setup

Consider the simple two layer neural network with ReLU activation (denoted by σ):

$$h(x) = \frac{1}{m} \sum_{i=1}^{m} a_i \sigma(b_i^T x)$$

Here, the weights are $a_i \in \mathbb{R}$ and $b_i \in \mathbb{R}^d$, and the output is normalized by $\frac{1}{m}$ where m is the hidden width, so that future analysis is simplified. Clubbing the weights into a parameter vector w_i for each hidden unit, we can rewrite this as:

$$h(x) = \frac{1}{m} \sum_{i=1}^{m} \Phi(w_i)(x)$$

where $\Phi : \mathbb{R}^{d+1} \to \mathcal{F}$ is a mapping that takes the weights associated with a single hidden unit and returns the corresponding scaled ReLU function. Moving to function spaces, we can clearly see the separability property of this two layer network:

$$h = \frac{1}{m} \sum_{i=1}^{m} \Phi(w_i)$$

Note that h here plays the same role as the discretized "function" $\mathbf{y}$ we saw in section 2.1.1. We can now assume a convex loss *functional* $R : \mathcal{F} \to \mathbb{R}^+$ and the empirical risk minimizer minimizes $G(W) := R(h(W)) = R\left(\frac{1}{m} \sum_{i=1}^{m} \Phi(w_i)\right)$, where W is a

vector of all the independent weights w_i. As the number of hidden units goes to infinity, we can replace the empirical average with an expectation over a measure on the weights:

$$F(\mu) := R\left(\int \Phi(w) d\mu(w)\right)$$

We can get back G from F by using a discrete sum of dirac measures $\mu = \frac{1}{m}\sum_{i=1}^{m} \delta(w_i)$.

Gradient Flows

Gradient descent on the objective G leads to the update equation:

$$W_{k+1} = W_k - m\gamma \nabla G(W_k)$$

In the limit of small step size, this discrete map approximates a continuous time gradient flow (the factor of m so that the infinite particle limit is consistent):

$$\dot{W} = -m\nabla G(W)$$

We can also write the gradient descent update in the proximal operator formulation:

$$W_{k+1} = \underset{W}{\operatorname{argmin}}\, G(W) + \frac{1}{2m\gamma} \|W - W_k\|_2^2 \tag{2.1}$$

This gradient update can be generalized to an arbitrary metric space by replacing the term $\|W - W_k\|_2^2$ with the metric distance $d(W, W_k)^2$. We will make use of this picture by looking at gradient flows in a metric that is natural for probability measures – the Wasserstein distance.

Wasserstein Distance

For simplicity, we will look at discrete measures with the same number of particles $\mu = \frac{1}{m}\sum_{i=1}^{m} \delta(w_i)$ and $\nu = \frac{1}{m}\sum_{i=1}^{m} \delta(v_i)$. The Wasserstein distance is the minimum

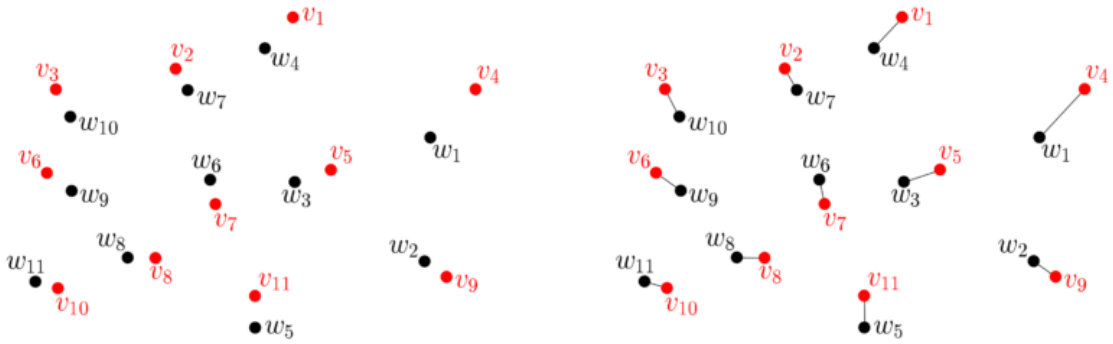


Figure 2.1: Wasserstein distance between two discrete probability measures. The lines indicate the optimal permutation σ. Notice that perturbing μ or ν slightly will not change the optimal permutation – this means that the Wasserstein distance is locally equal to a decoupled sum of Euclidean distances. Image credits [7].

sum of Euclidean distances required to move the particles in μ to ν:

$$d_W(\mu, \nu) = \min_{\sigma} \frac{1}{m} \sum_{i=1}^{m} \left\| w_i - v_{\sigma(i)} \right\|_2^2$$

Where the minimum is taken over all permutations $\sigma : \{1, \ldots, m\} \to \{1, \ldots, m\}$.

The key idea is to notice that if μ or ν is perturbed slightly, the optimal permutation σ does not change. This means that, locally, the Wasserstein metric behaves like a sum of Euclidean distances. We can rewrite Eq. 2.1 by separating out each particle's weight as follows:

$$W_{k+1} = \operatorname*{argmin}_{W} G(W) + \frac{1}{2m\gamma} \sum_{i=1}^{m} \left\| w_i - w_i^{(k)} \right\|_2^2$$

Now, as the step size $\gamma \to 0$, we can use the intuitive approximation derived above and replace the sum of Euclidean distances with a Wasserstein distance between measures:

$$\mu_{k+1} = \operatorname*{argmin}_{\mu} F(\mu) + \frac{1}{2m\gamma} d_W(\mu, \mu_k)$$

Where the minimization is done over discrete measures with a fixed number of particles m. Therefore, in the limit of small step sizes $\gamma \to 0$, and infinite particles $m \to \infty$, the Euclidean gradient flow of G approaches the Wasserstein gradient flow of F. This is an intuitive derivation, and rigorous details can be found in [9].

Global Convergence

Now, the main result of [9] is that the Wasserstein gradient flow of F converges to a global minimum (if it converges) under some regularity assumptions:

- **Homogeneity:** Each independent functional Φ should be homogenous in the weight w.
- **Initialization:** The initial locations of the particles should be such that there is sufficient positive mass in all directions. This is satisfied by the usual Gaussian initialization. Comparing this with standard initializations that lead to NTK dynamics, we notice that the additional $\frac{1}{m}$ factor in the definition of the network output h can be brought into the initialization of the weights, and we get that the initialization is much smaller with standard deviation going as $\frac{1}{m}$ compared to $\frac{1}{\sqrt{m}}$ in Lecun initialization.

The details of the proof of this result are quite involved, and can be found in the paper [9].

2.2.2 Polyak-Lojasiewicz condition for Non-Convex Optimization

The central idea behind the NTK theory is to show that, in the infinite width limit, the non-convex training dynamics of a neural network approach a linear approximation. Using this, we can obtain linear convergence guarantees for gradient descent because the resulting optimization problem is convex. However, [20] shows that we can obtain these linear convergence guarantees for overparameterized models in a regime where the NTK still changes and the optimization landscape is provably non-convex. This is done by first showing that a slightly less strict requirement called the Polyak-Lojasiewicz condition (PL-condition) is satisfied by neural networks of finite width before entering the NTK regime. This condition then implies a linear convergence rate for gradient descent.

PL Condition

A loss function $\mathcal{L}(w)$ is μ-PL on a domain S if it satisfies:

$$\frac{1}{2}\left\|\nabla\mathcal{L}(w)\right\|^2 \geq \mu(\mathcal{L}(w) - \mathcal{L}(w^*)) \quad \forall w \in S \tag{2.2}$$

Where w^* is a global minimizer. We additionally assume that the loss $\mathcal{L}(w)$ has an L-Lipschitz gradient:

$$\mathcal{L}(y) \leq \mathcal{L}(x) + \nabla\mathcal{L}(x)^T(y-x) + \frac{L}{2}\|y-x\|^2$$

With these two assumptions, we can obtain linear convergence of gradient descent. Assuming we use a step size of $\frac{1}{L}$:

$$w_{k+1} = w_k - \frac{1}{L}\nabla\mathcal{L}(w_k)$$

We get using eq. 2.2,

$$\mathcal{L}(w_{k+1}) \leq \mathcal{L}(w_k) - \frac{1}{L}\nabla\mathcal{L}(w_k)^T\nabla\mathcal{L}(w_k) + \frac{L}{2}\left\|\frac{1}{L}\nabla\mathcal{L}(w_k)\right\|^2$$

$$\implies \mathcal{L}(w_k) - \mathcal{L}(w_{k+1}) \geq \frac{1}{2L}\|\nabla\mathcal{L}(w_k)\|^2 \geq \frac{\mu}{L}(\mathcal{L}(w_k) - \mathcal{L}(w^*))$$

Now we have a lower bound on the reduction in loss in each step that is proportional to the current excess loss. This is exactly the criterion for linear convergence, and we have obtained the desired result.

Uniform Conditioning

To show that certain neural network architectures satisfy the PL condition, we use the concept of uniform conditioning. For a given network function $F(w)$ where w is the parameter vector, the corresponding NTK is $K(w) = \nabla F(w)\nabla F(w)^T$. We say F is μ-uniformly conditioned on a domain S if

$$\lambda_{min}(K(w)) \geq \mu \quad \forall w \in S \tag{2.3}$$

Now, the main utility of this concept is that the uniform conditioning of $F(w)$ implies the PL condition for the square loss $\mathcal{L}(w) = \frac{1}{2}\|F(w) - y\|^2$, if the global minimum is

0 (i.e., $\mathcal{L}(w^*) = 0$). The proof is straightforward:

$$\begin{aligned}
\frac{1}{2}\left\|\nabla\mathcal{L}(w)\right\|^2 &= \frac{1}{2}\left\|(F(w)-y)\nabla F(w)\right\| \\
&= \frac{1}{2}(F(w)-y)K(w)(F(w)-y)^T \\
&\geq \frac{1}{2}\lambda_{min}(K(w))\left\|F(w)-y\right\|^2 \\
&= \lambda_{min}(K(w))\mathcal{L}(w) \geq \mu\mathcal{L}(w) \quad \text{using Eq. 2.3}
\end{aligned}$$

Intuitively, uniform conditioning implies that the NTK is locally positive definite with a certain minimum eigenvalue. This means that the optimization of squared loss using the **linearized** function at any point in the domain S is μ-strongly convex. This is sufficient for gradient descent to have a linear convergence rate. Note that this **need not imply** strong convexity of the true (not approximated) objective. In fact, the authors of [20] show that overparameterized systems are essentially non-convex – which means that the objective is not convex in any neighborhood of global minima. This can also be stated as – the manifold of global minimizers is non-linear and has non trivial curvature.

Building on this story, [20] shows that common overparameterized neural network architectures satisfy the uniform conditioning property around their initializations as the network width is increased. Of key importance in this result is that the required width is much lesser than the quantitative results required to enter the NTK regime. They empirically confirm that networks operating in the PL regime have significantly changing NTKs and do not follow a linearized optimization trajectory.

2.3 Deep Linear Networks

Another avenue of research where interesting theoretical results have been arising is that of **deep linear neural networks**. These neural network architectures are almost the same as standard fully connected architectures with one major difference – there are no non-linear activations. The network output can be written simply as (where d is depth):

$$f(x) = W_d \ldots W_3 W_2 W_1 x$$

At first blush, this seems rather uninteresting because a composition of many linear

layers can be reduced to a single linear layer. While the end-to-end function is indeed linear, training this architecture using gradient descent is by no means trivial. Even for the simple case of $d = 2$, the square loss objective is non-convex in the weights W_i and training dynamics of gradient descent is highly non-linear. The exact form of the differential equations of gradient flow on this architecture with the square loss can be found in [22]. Exact solutions for a depth of 2 are also presented in the paper.

More recent works [3] have obtained quantitative convergence rates for arbitrary depth as well. Another interesting line of work is in analyzing the implicit bias of gradient descent on deep linear architectures. Specifically, multiple works [4, 16] have identified that gradient descent prefers solutions where the end-to-end linear mapping is of low rank. This bias is seen to strengthen with more depth. Theoretical analysis has shown that this implicit regularization and cannot by captured by other mathematical formulations like the nuclear norm or Schatten quasi-norms [16].

From a high level view, studying deep linear networks seems uninteresting because the end-to-end function is still linear, but the non-trivial learning dynamics of these architectures has revealed hard and interesting questions and serves as motivation to pursue other architectures with deep linear compositions.

CHAPTER 3

Setting of the Case Study

The aim of this case study is analyze the next level of complexity from the basic one hidden-layer neural network (sometimes called a two-layer network) for which significant theoretical progress has been made in terms of optimization and convergence beyond the NTK regime (see background section 2.2.1).

3.1 Dataset

The dataset which we will use for all experimental studies is a minimal extension on the basic 1-D learning dataset. In the case of 1-D learning, the target function $g^* : \mathbb{R} \to \mathbb{R}$ is usually assumed to possess some smoothness properties (for example take $g^*(x) = \sin(\pi x)$, and the data (x_i, y_i) is generated by drawing x from an assumed input distribution (commonly Gaussian), and y is obtained by applying the true target function with some noise: $y_i = g^*(x_i) + \epsilon_i$. However, in this case study, we restrict ourselves to the noiseless case, so $y_i = g^*(x_i)$.

To lift this dataset to higher dimensions, we can simply project the high dimensional input onto a fixed 1-D line and apply the same target function g^*. The dataset $(\mathbf{x_i}, y_i)$ where $\mathbf{x_i} \in \mathbb{R}^d$ is now obtained using the following generation process:

$$y_i = g^*({\mathbf{w}_*}^T \mathbf{x_i}) \tag{3.1}$$

Where $\mathbf{w}_* \in \mathbb{R}^d$ defines the direction of the line onto which the input is projected. We additionally assumed without loss of generality that $\|\mathbf{w}_*\| = 1$ (any scaling in $\mathbf{w}_*$ can be transferred to g^*, and its Lipschitz constant will correspondingly change). The end-to-end target function $f^* : \mathbb{R}^d \to \mathbb{R}$ is visualized in Fig. 3.1 for a simple 2-D example.

In the 1-D learning case, we used a standard Gaussian $x \sim \mathcal{N}(0, 1)$ as the input data distribution. In the higher dimensional setting, we use $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as the input dis-

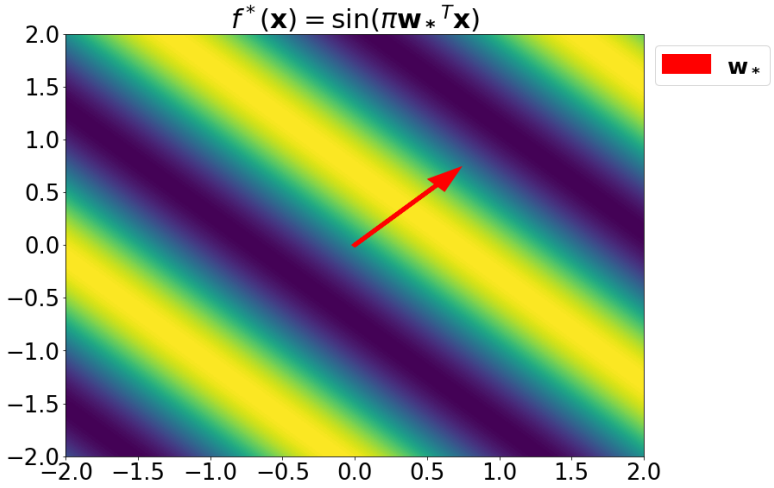


Figure 3.1: An example of the end-to-end target function f^* with $d = 2$, and $g^*(x) = \sin(\pi x)$. The projection is performed along $\mathbf{w}_* = \frac{1}{\sqrt{2}}[1, 1]^T$. Notice that this class of target functions forms a very simple subset of possible functions from $\mathbb{R}^d$ to $\mathbb{R}$.

tribution. Since we have chosen $\|\mathbf{w}_*\| = 1$, the input to the 1-D learner still follows a standard Gaussian distribution, $\mathbf{w_*}^T\mathbf{x} \sim \mathcal{N}(0, 1)$.

3.2 Model Architecture

As stated earlier, we motivate this case study by taking existing 1-D learner architectures and increasing the complexity of the problem by lifting to higher input dimensions in the simplest way possible. Consider the two layer neural network 1-D learner of width m studied in background section 2.2.1: $g_\theta(x) = \mathbf{v}^T\sigma(\mathbf{u} \cdot x + \mathbf{b})$, where $\mathbf{u} \in \mathbb{R}^m$ is the hidden layer weight vector with bias $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{v} \in \mathbb{R}^m$ is the output layer weight. The non-linear activation σ is ReLU. We can easily lift this model to higher dimensions in the same way we did for the dataset, by adding an additional direction $\mathbf{w} \in \mathbb{R}^d$ onto which the input is first projected:

$$f(\mathbf{x}) = g_\theta(\mathbf{w}^T\mathbf{x}) \tag{3.2}$$

$$= \mathbf{v}^T\sigma(\mathbf{u}\mathbf{w}^T\mathbf{x} + \mathbf{b}) \tag{3.3}$$

Note that this lifting could be done with any black box 1-D learner model g_θ, but we choose to stick to the specific case of the two layer network. The parameters of f are $\mathbf{w}$, the projection direction, and θ, the parameters of the 1-D learner. From Eq. 3.3, we

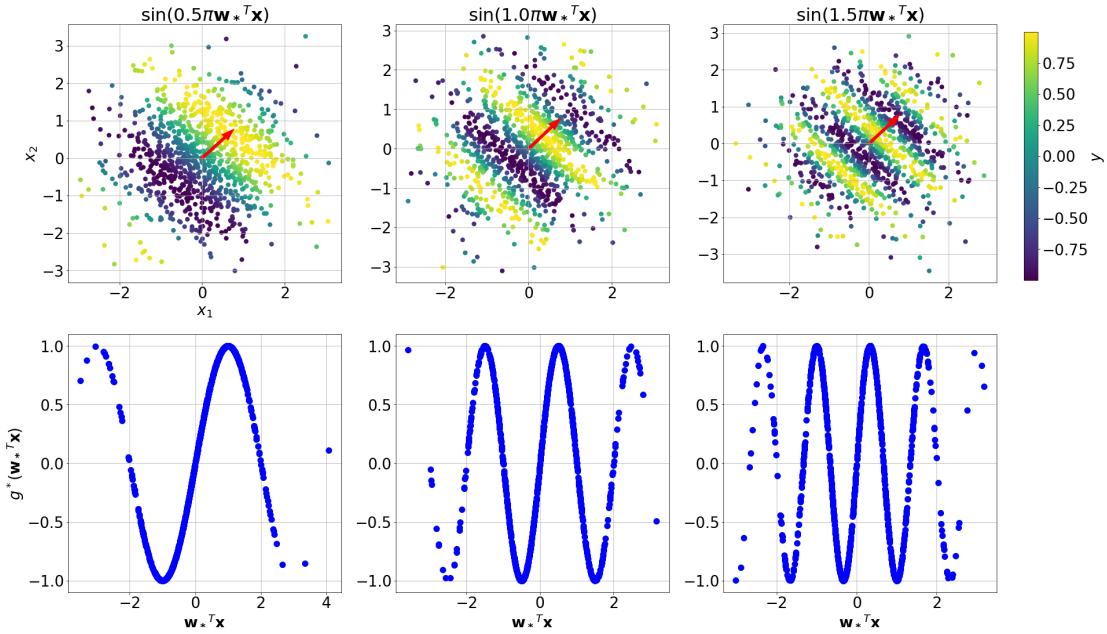


Figure 3.2: A scatter plot of sample train datasets for three frequencies of $\sin$, with $d = 2$. The input data distribution is a standard 2-D Gaussian, $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The bottom plots show the data projected onto the $\mathbf{w}_*$ direction, along with their corresponding y values.

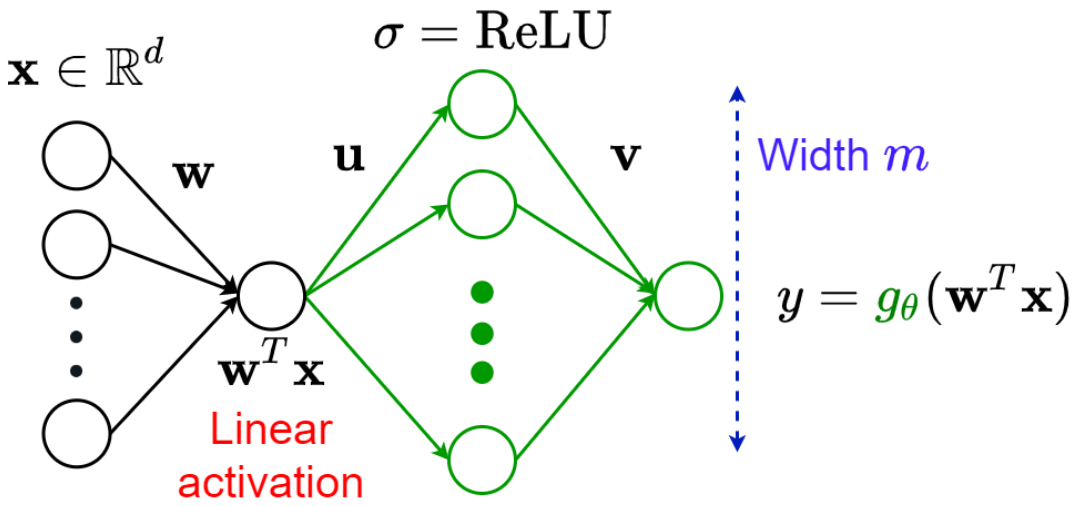


Figure 3.3: The architecture of the model used in the case study. It is obtained by lifting the basic two layer neural network architecture for 1-D learning to higher dimensions by adding a linear input layer of width 1, instead of directly increasing the input dimension of the first layer. This represents the simplest three-layer architecture, and has significant advantages in terms of what can be probed to better understand the learning process.

can purport to directly parameterize f as a two layer network without first projecting onto $\mathbf{w}$ by combining $\mathbf{u}$ and $\mathbf{w}$ into one linear layer. However, as we saw in 2.3, deep linear architectures cannot be trivially reduced to a single linear layer as their training dynamics under gradient descent are more complex. Additionally, this formulation of the model aligns with the data generation process. This provides us with a useful advantage – we know beforehand the ideal values of each component of the model, i.e., we know that the best value for $\mathbf{w}$ is $\mathbf{w}_*$, and the best function for the 1-D learner is g^*. This extra knowledge allows us to create new metrics (details in section 4.2.1) that can probe into the intermediate layers of the deep network and offer new information about the learning dynamics that is not normally available in other deep learning settings. For instance, we have no information about which weight values are better for the first or second layer while training a convolutional network on the MNIST dataset.

Finally, we note (see Fig. 3.3) that this architecture is effectively the simplest three-layer deep architecture, with the first layer having a linear activation and a bottleneck width of 1. This simplicity is what affords the extra flexibility in measuring the behavior of the network as it learns by gradient descent. We will get into more details in later sections.

CHAPTER 4

Empirical Analysis

In this chapter, we explore a number of hypotheses that deal with the learning dynamics of the setting described in chapter 3. As with the setting in 1-D learning, we train the complete network using stochastic gradient descent on the square loss $L = \frac{1}{2}\sum_i \|y_i - f(\mathbf{x_i})\|^2$. For all the experiments in this chapter, we fix some common hyperparameters unless explicitly stated otherwise:

- Input dimension $d = 10$.
- Batch size = 32.
- Optimizer: SGD (no momentum), with learning rate $5 \cdot 10^{-4}$.

4.1 1-D Learning Baselines

Before we analyze the experiments on the three-layer architecture, we first establish some results and baselines for the 1-D learner architecture which we use in the case study. As stated earlier in section 3.2, we use a two-layer neural network for the 1-D learner g_θ. The theory described in background section 2.2.1 provides asymptotic guarantees on the convergence of this architecture to the global minimum as the hidden width goes to infinity. Empirically, however, we observe that a finite width is sufficient to achieve successful generalization, where success is measured by using validation loss. For simplicity, we shall stick to using $g^*(x) = \sin(\nu\pi x)$ as our target function, with the frequency of the sin varying from 0.5 to 1.5. The input data distribution is a standard Gaussian, $x \sim \mathcal{N}(0, 1)$, and no noise is added in the data generation process.

Fig. 4.1 presents the results of multiple runs of 1-D learning with different random initializations for different target functions and train dataset sizes. To get a sense for how the learned function looks for different values of the MSE loss, see Fig. 4.2. It is clear that 1-D learning succeeds for all the frequencies, however a measurably worse performance is seen with increasing frequency of the target function.

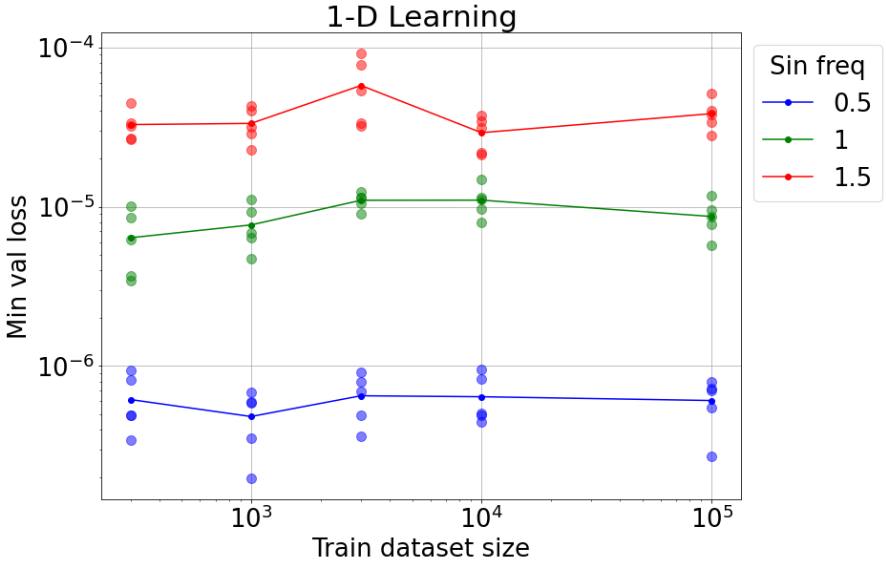


Figure 4.1: 1-D learning using the two-layer neural network architecture analyzed in section 2.2.1. Each point corresponds to a single run, and the lines join the averaged loss values. All runs used the same hyperparameters, with a hidden width of 1000 units. Although all frequencies are "successfully" learned by the 1-D learner, higher frequencies do have measurably worse performance in terms of squared loss. Additionally, we observe that increasing train dataset size does not seem to noticeably affect performance.

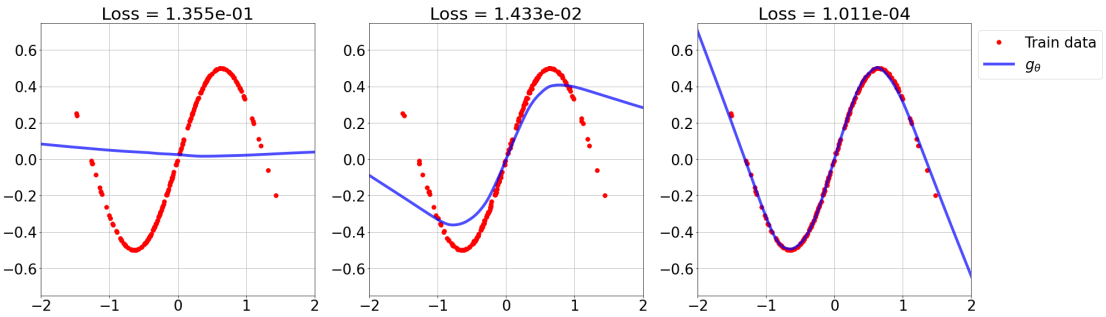


Figure 4.2: Visualizing learned functions with different mean squared errors.

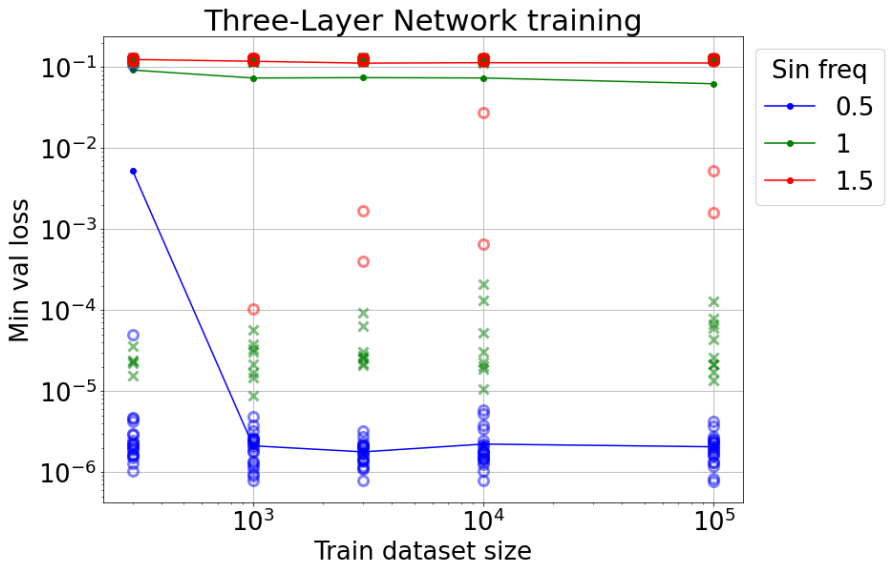


Figure 4.3: The results of training the three-layer network on data generated as described in section 3.1. The lines correspond to the averaged loss value of 20 runs for size of the training dataset. Each individual run is also plotted by markers. A number of runs for 1 and 1.5 Hz heavily overlap around a loss of 10^{-1}, which make the average curves higher.

4.2 Comparison with 1-D Learning

Given that we have a 1-D learner g_θ that is capable of successfully learning the target function g^*, a natural question to ask is if this capability extends to our setting where we need to learn an additional linear mapping before the 1-D function. We formalize this question as hypothesis 1.

Hypothesis 1. *If the 1-D learner g_θ can learn g^* with high probability, then the model (Eq. 3.3) $f(x) = g_\theta(\mathbf{w}^T x)$ can learn the composed target function f^* with high probability as well.*

Intuitively, one can argue for both possible results. Learning a linear mapping by gradient descent is known to be "easy" in isolation, so composing it with 1-D learning, a task that is also solvable using the learner we have chosen, should also have a high success rate. However, as we saw in section 2.3, composing linear layers can lead to highly non-convex and non-trivial dynamics, and the 1-D learning theory need not hold by any means in this setting.

This hypothesis is easily tested, by simply running stochastic gradient descent on the three-layer architecture using sampled data. The results are presented in Fig. 4.3.

The results of Fig. 4.3 show that hypothesis 1 does not seem to hold water. While lower frequencies like 0.5 Hz are successfully learnt, higher frequencies do not learn with a very high probability. Having a 1-D learner capable of learning g^* is clearly not sufficient to learn the end-to-end function f^* in higher dimensions. We also observe an interesting phenomenon from the individual run markers – the loss values separate into two modes, corresponding to "success" and "failure" of learning. To better understand this, we can use an additional metric that is made possible due to the setting of our case study – cosine similarity.

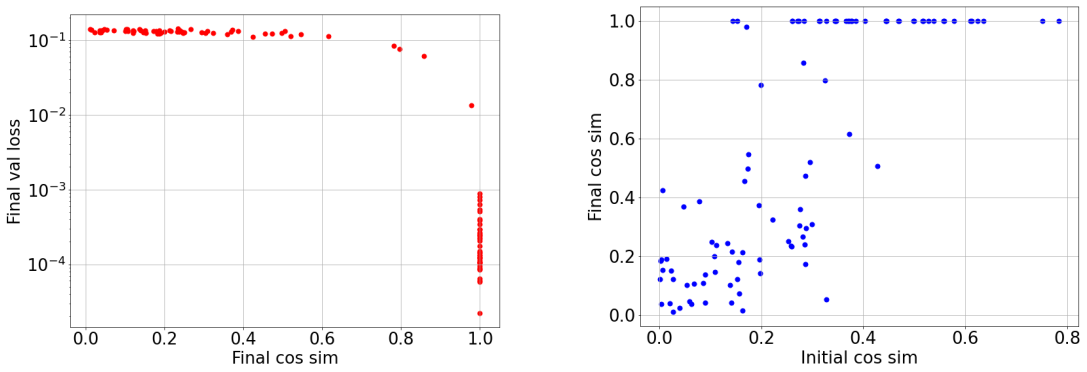
4.2.1 Cosine Similarity

Since the model which we use to learn has the same structure as the target function until the 1-D learner, we can attempt to directly compare the learned projection direction $\mathbf{w}$ with the true direction $\mathbf{w}_*$. The architecture is such that an arbitrary scaling can be transferred between $\mathbf{w}$ and the learnt 1-D function g, so we only care about the difference in **directions** between $\mathbf{w}$ and $\mathbf{w}_*$. This is perfectly measured using the *cosine similarity* metric:

$$\text{CosSim}(\mathbf{w}, \mathbf{w}_*) = \frac{{\mathbf{w}_*}^T \mathbf{w}}{\|\mathbf{w}\|_2 \cdot \|\mathbf{w}_*\|_2} \tag{4.1}$$

Along with the scaling, a factor of -1 can also be transferred between $\mathbf{w}$ and g (learning $-\mathbf{w}_*$ is also sufficient for success, as the 1-D learner can then learn $-g$). So, we can further refine the metric and only look at the *absolute value* of the cosine similarity.

From Fig. 4.4a, we clearly observe that the "successful" runs with loss values in the range of 10^{-3} to 10^{-4} have perfectly learnt the direction of $\mathbf{w}_*$ with a CosSim of 1. The failed runs have a CosSim less than 1. Another interesting use of this metric is the ability to predict learning from properties of the initialization. Fig. 4.4b shows that initializations with a CosSim closer to 1 are more likely to succeed. The initialization used for $\mathbf{w}$ is the standard LeCun initialization, so the CosSim is not uniformly distributed from 0 to 1, and instead concentrates around 0. The skew increases with an increase in input dimension. Intuitively, two random Gaussian vectors in high dimensional Euclidean space are more likely to be orthogonal as the dimension increases.



(a) A scatter plot of 100 runs on the same target function ($\sin(0.7\pi x)$), with all hyperparameters identical. The only difference is in the random initializations. The separation of loss into two modes can be explained by the absolute cosine similarity of the learned $\mathbf{w}$ with $\mathbf{w}_*$.

(b) The same 100 runs are used as in the left plot. The initial CosSim is a strong predictor of success – a higher initial CosSim is more likely to result in a final CosSim of 1.

Figure 4.4: Cosine Similarity as a metric to probe learning

We can modify the initialization scheme slightly to better understand the impact of the CosSim. Instead of using a standard Gaussian initialization, we restrict the initial weight $\mathbf{w}$ to lie in a cone at a specific angle to the true weight $\mathbf{w}_*$. This essentially fixes the initial CosSim to a value of our choice. This scheme is implemented by taking a linear combination of $\mathbf{w}_*$ and a random orthogonal vector. The weights in the linear combination can be controlled to yield the desired CosSim. The random orthogonal vector is generated by subtracting the component along $\mathbf{w}_*$ from a standard random normal vector. The results of running the training algorithm with this modified initialization scheme is shown in Fig. 4.5.

By monitoring CosSim during the training of the network, we can probe into the characteristics of how learning occurs. We observe interesting phenomenon using this technique, as we shall see in the next section. A final point to note is that most other deep learning settings do not have this ability to directly measure a metric using weight vectors in the network. Usually, the weights themselves have little information to offer in terms of learning performance. The advantage of the setting in this case study is that when the learning process is successful, it consistently recovers the true weight used to generate the data, upto scaling factors. This ability allows to test interesting hypotheses and conduct experiments that are normally not possible in a deep learning setting.

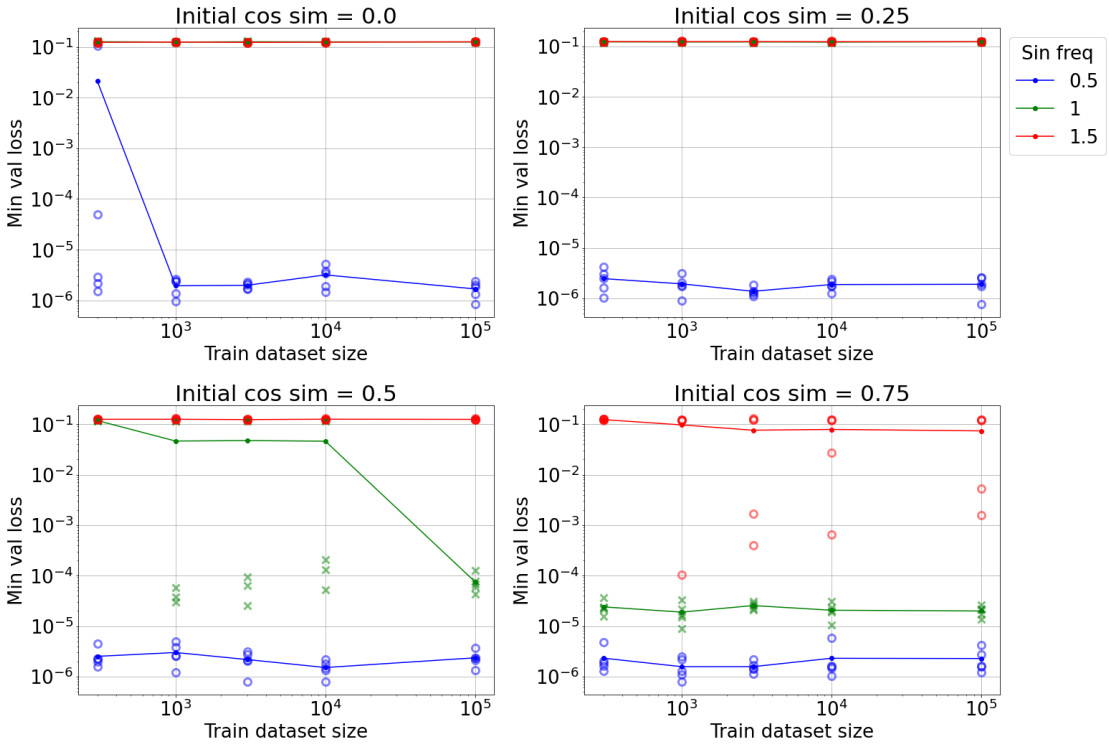


Figure 4.5: Training with the fixed initial CosSim initialization scheme. Higher frequencies like 1 Hz are learnt successfully if the initialization is a favorable one with high initial CosSim. However, frequencies like 1.5 Hz still do not show as successful learning as in the 1-D case, so hypothesis 1 is still incorrect.

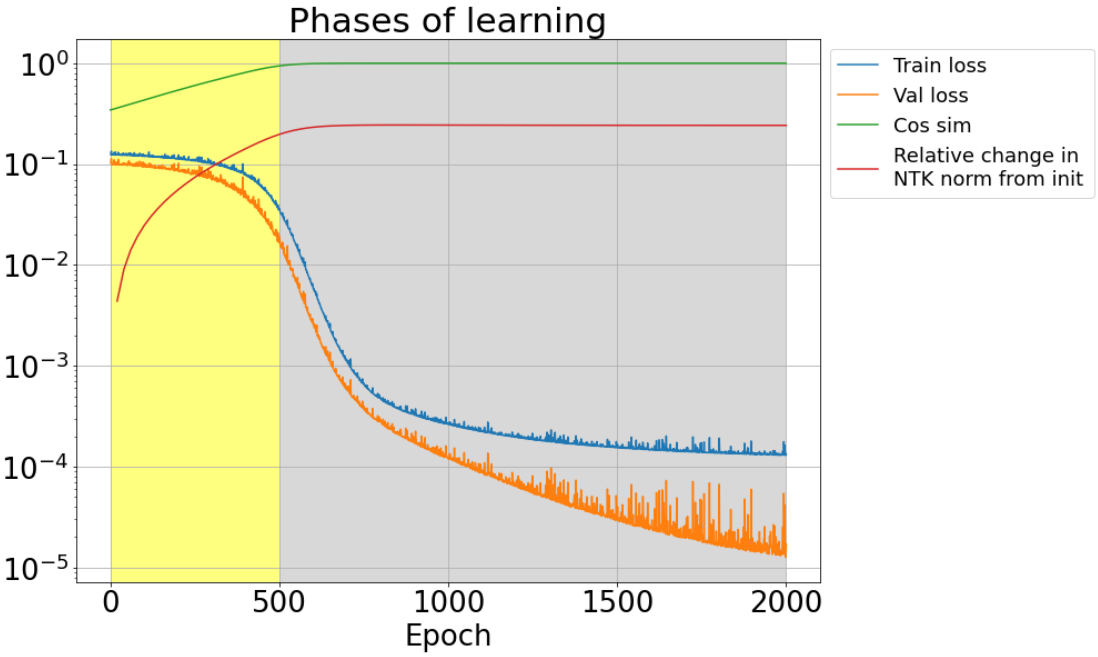


Figure 4.6: Observe the different phases of training. Initially, the cosine similarity grows towards 1, while the loss barely changes (until epoch $\sim$ 500). In this phase, the NTK also changes significantly from initialization. Once the cosine similarity has saturated, we observe a sharp drop in loss. After this phase, the NTK change has also saturated, and the loss decays exponentially as expected in the NTK regime.

4.3 Phases of Learning

In search of a theoretical explanation for the success in training, we first present the following hypothesis:

Hypothesis 2. *Successful learning can be explained by the NTK theory as described in section 2.1.*

Fig. 4.6 shows the different metrics for a successful run of training using a target function of $g^*(x) = \sin(0.5\pi x)$. In addition to the metrics discussed so far, another quantity is also plotted – the relative change in the norm of the NTK from initialization. Precisely, if we let t denote the epoch index, then we plot the quantity

$$\text{Relative change in NTK} = \frac{\|\text{NTK}(t) - \text{NTK}(0)\|_F}{\|\text{NTK}(0)\|_F}$$

The plot clearly shows that the NTK significantly changes in the early stages of learning. The NTK does not stay constant throughout training, and this immediately refutes hypothesis 2. However, we do observe that the NTK remains roughly constant in the

later stages of learning. This suggests that learning occurs in multiple phases, and the NTK theory can explain the later part of learning.

Previous works on analyzing gradient descent in deep learning have also observed this phenomenon where learning occurs in multiple, distinguishable phases. For instance, [23] explores deep networks from an information theoretic point of view, and identifies two phases of stochastic gradient descent – drift and diffusion. They provide empirical results that seem to agree with this claim. More recently, [21] identifies a common structure of multi-phase learning among different problems in which each phase learns functions of increasing complexity. They develop interesting empirical metrics to further substantiate their observations. Even more recently, [18] identifies two phases of learning when using large learning rates and terms this behavior the catapult mechanism. For a particular range of large, stable learning rates, the loss initially increases exponentially. However, after this initial period of "catapulting", the loss begins to decrease as per the NTK dynamics discussed in background section 2.1. The advantage of this setting is that the resulting minima are usually flatter. The authors prove this separation of learning into two phases for an extremely simplified linear model, and also empirically observe similar results in more complex architectures.

In the context of our case study, we also observe the separation of learning into two phases based on the variability of the NTK. Here we see another benefit of having a metric like the CosSim that allows us to probe into the inner workings of the network during learning. The CosSim metric reveals an additional interesting observation – the saturation of the NTK coincides with the CosSim reaching 1. Following this line of inquiry, we present the following hypothesis:

Hypothesis 3. *Learning occurs in two distinct phases, where* $\mathbf{w}$ *is learnt in the first phase (along with* g*) with a changing NTK, and the second phase involves only improvements in* g *with a constant NTK.*

The results presented in Fig. 4.6 support hypothesis 3 as well. We observe that the NTK significantly deviates from initialization in tandem with the CosSim monotonically increasing to 1. Once the CosSim has reached 1, i.e., $\mathbf{w}$ has successfully been learnt, the NTK saturates, and the loss decreases exponentially as expected in the NTK regime. The continued decrease in loss can be attributed to fine improvements in g as $\mathbf{w}$ is already learnt by the end of phase 1. To further support this hypothesis, the series of plots

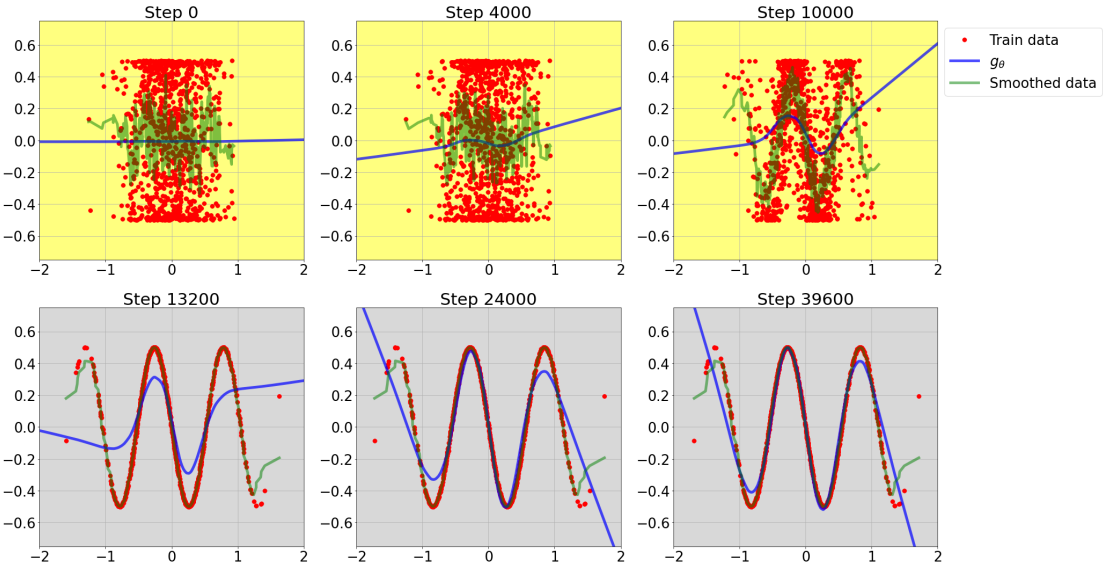


Figure 4.7: A series of frames for a single run of training. We visualize the functions in the domain of the 1-D learner component. The x-component of each training datapoint (plotted as red dots) is the value of its projection onto the current value of the weight $\mathbf{w}$, and the y-component is the true y value. The blue curve is a plot of the current function g_θ. The green curve is a Gaussian-smoothed version of the red points, as a baseline simple 1-D approximator.

in Fig. 4.7 visualize the different components of the network as learning progresses.

The frames presented in Fig. 4.7 further provide evidence supporting hypothesis 3. Once $\mathbf{w}$ is learnt by step ~ 13200, we observe that the red points stop moving. This indicates that $\mathbf{w}$ has been learnt successfully and is no longer changing. However, the loss still decreases as improvments in the 1-D learner g_θ is still occuring, as can be seen by minor variations in the blue curve. An additional observation by comparing the blue curve with the green Gaussian-smoothed curve is that the 1-D learner fits the low frequency components of the red datapoints much faster than $\mathbf{w}$ itself changes. This is a vague and intuitive observation that suggests a possible direction for a precise theoretical explanation.

The duration of the first phase is quite variable as well. Fig. 4.9 shows another run of the same architecture in which the initial phase lasts 3 times longer than the previous example in Fig. 4.6. This is similar to the behavior of SGD escaping saddle points [11]. Additionally, the train and validation loss metrics show essentially no change during the first phase, even though it is much longer in this case. This behavior has been observed in practical scenarios of deep learning as well, and is usually identified as a saddle point in the optimization objective [12]. They are also usually attributed as the cause

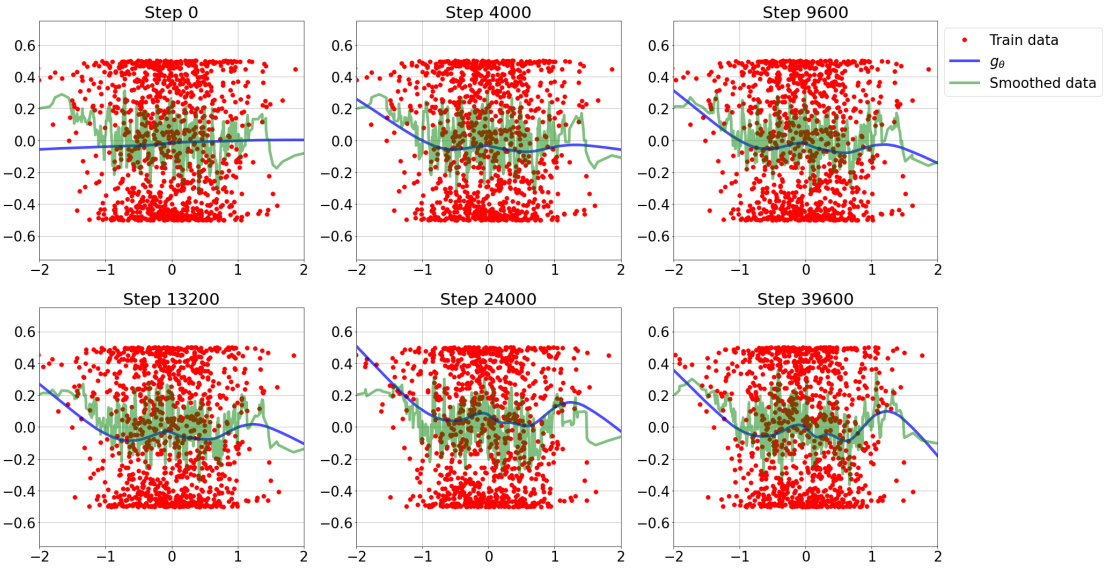


Figure 4.8: A series of frames as training progress for a run in which learning fails. The optimization gets stuck in a local minimum and $\mathbf{w}_*$ is not recovered.

for the presence of multiple phases of learning. In our case, we additionally observe that the CosSim still shows a monotonic increase with what seems like an exponential trend. These observations could serve as useful guides for future work in developing a theoretical explanation of the observed learning dynamics.

4.4 Experiments with Parameter Control

Since we know the *true parameters* required to achieve a low validation loss on the dataset, at least for $\mathbf{w}$, we can experiment with training with some control over different parameters. This is unlike most deep learning settings where the exact values of the true parameters or weights is not known. We can either fix or initialize parts of the network to their true values and train the remaining or whole network in an attempt to decompose the learning dynamics. Here we present 4 variations following this approach.

4.4.1 Fixing $\mathbf{w}$ to $\mathbf{w}_*$

If we force $\mathbf{w}$ to be equal to $\mathbf{w}_*$ and then train only the 1-D learner, our problem trivially reduces to exactly the 1-D learning problem. So this case has already been studied.

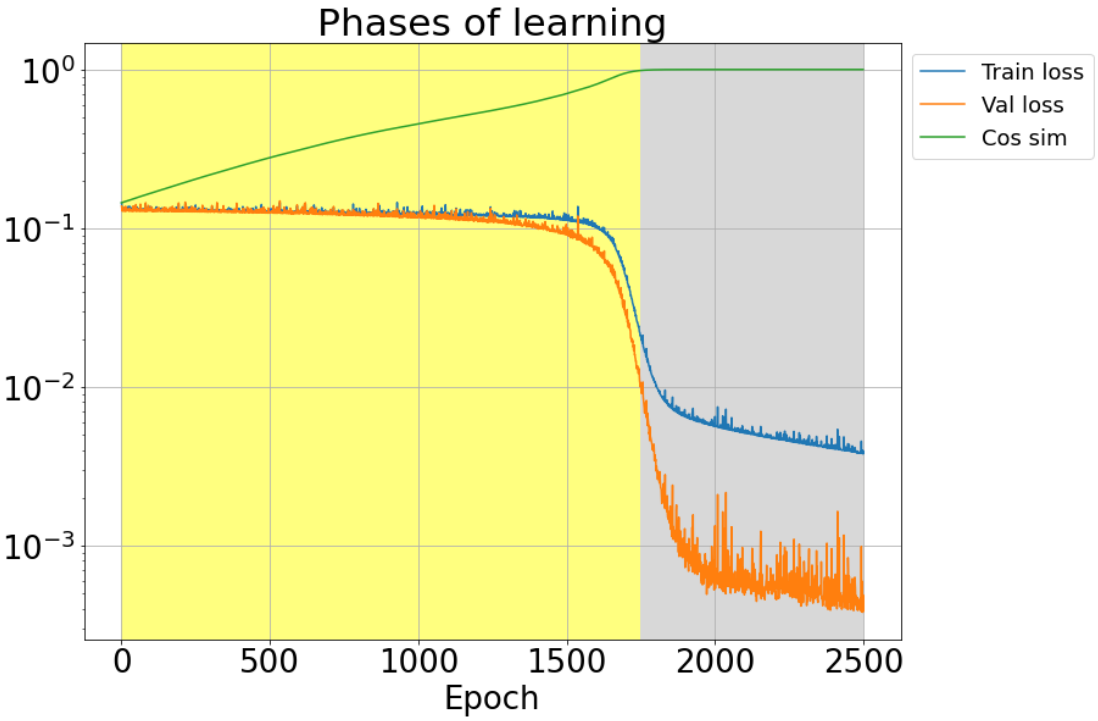


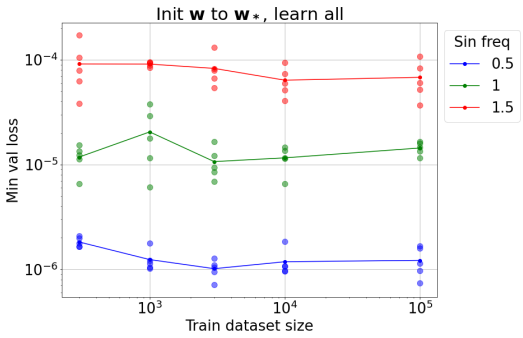
Figure 4.9: Another instance to show the different phases in learning. In this run, the first phase lasts for much longer ($\sim$ 1750 epochs), during which the train and validation losses barely change. However, the CosSim is seen to monotonically increase towards 1, when the phase ends.

4.4.2 Initializating $\mathbf{w}$ to $\mathbf{w}_*$

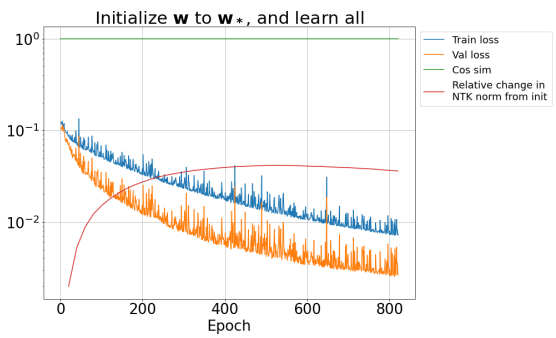
However, if we instead just *initialize* $\mathbf{w}$ to $\mathbf{w}_*$ and then train the *whole network*, it is not immediately obvious that we would observe the same results as in the 1-D learning case, because the parameter $\mathbf{w}$ can still change. We formalize this as hypothesis 4:

Hypothesis 4. *Learning the three-layer network with* $\mathbf{w}$ *initialized to* $\mathbf{w}*$ *and training all parameters behaves exactly like 1-D learning, and* $\mathbf{w}$ *does not move from its initialization.*

The results presented in Fig. 4.10 strongly support hypothesis 4. The loss values of all runs are "success" values and Fig. 4.10a is extremely similar to the 1-D learning results in Fig. 4.1. Fig. 4.10b shows the metrics of one single run in this setting. The CosSim stays at 1 throughout, so only g_θ is being learnt, and this amounts to 1-D learning. The NTK still changes significantly throughout training, and the NTK theory is insufficient to explain the dynamics of learning. Instead, the optimal transport theory in section 2.2.1 captures the richer dynamics of this setting, as it does in the 1-D learning case. A plausible explanation for the results that we observe is using the observation



(a) The success rate of initializing $\mathbf{w}$ to $\mathbf{w}_*$ is more or less identical to that of 1-D learning.



(b) A single run of this parameter control scheme. Notice that no distinguishable phases are identifiable, but the NTK still changes, so NTK theory is insufficient to explain the dynamics.

Figure 4.10: Results of training the whole network after initializing $\mathbf{w}$ to $\mathbf{w}_*$. This is identical to fixing the initial CosSim to 1.

pointed out in section 4.3 – the rate at which g_θ learns is much faster than the rate at which $\mathbf{w}$ learns. With this assumption, the 1-D learner fits the ideal function much faster than $\mathbf{w}$ could move away from its initialization at $\mathbf{w}_*$. Again, these are qualitative and speculative statements that can guide future work on developing precise theoretical explanations.

4.4.3 Fixing g to g*

In the previous sections, we controlled $\mathbf{w}$ and did not interfere with the initialization or training of the 1-D learner. Now, we look at the scenario where we still initialize $\mathbf{w}$ randomly (or with a fixed initial CosSim), and control g_θ. We first analyze the case where we know the true function g^*, and only need to learn the direction of projection. One strategy is to use the same architecture we have been using till now, and replace the 1-D learner with a fixed function g^*. We have effectively fixed the 1-D learner to its ideal value. This setting can be thought of as learning a simple one-layer network with a non-linear activation function g^*. Intuitively, one would expect this setting to be easier than the general setting of learning both $\mathbf{w}$ and g_θ.

Hypothesis 5. *The success rate of learning* $\mathbf{w}$ *should improve if we replace the 1-D learner with a fixed function equal to* g^*.

Surprisingly, from Fig. 4.11, we observe that this is not the case. Hypothesis 5 does not hold. In fact, by fixing g to g^*, the problem has become noticeably harder as even a

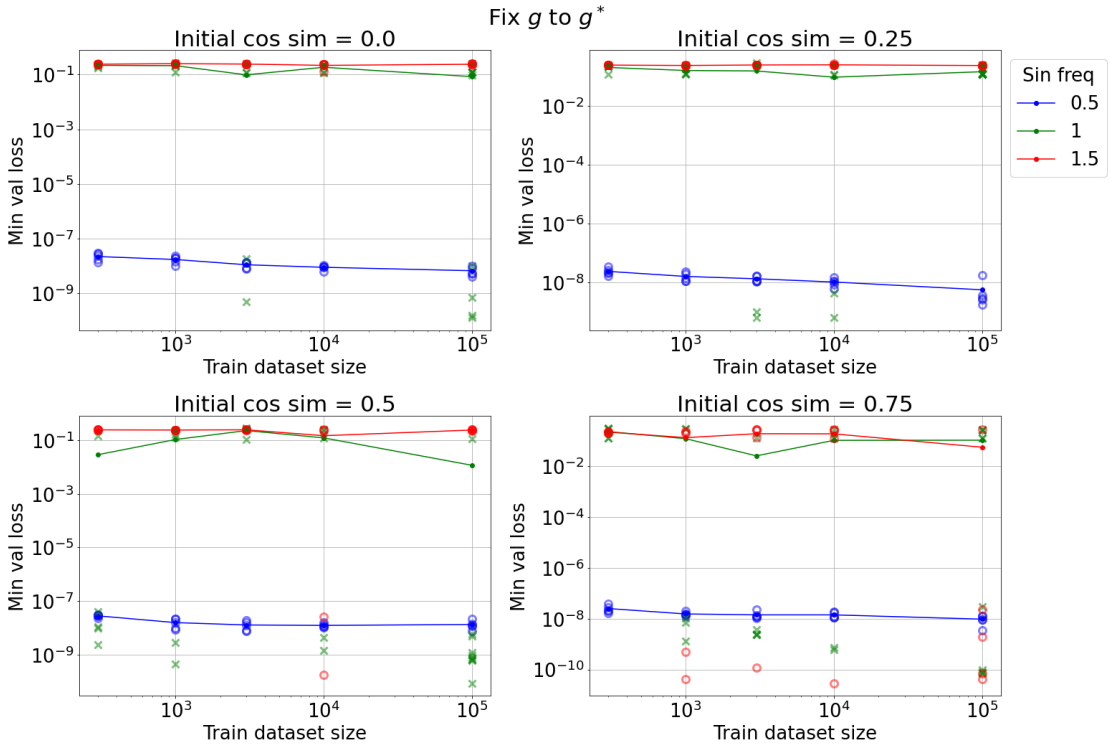


Figure 4.11: Training only $\mathbf{w}$ with g_θ fixed to the true function g^*. Compare these results with Fig. 4.5, the normal case where both components are initialized randomly and learnt together.

frequency of 1 Hz does not succeed with a high probability while it used to succeed in the case of learning both g_θ and $\mathbf{w}$. To understand why such a counter intuitive result is plausible, we present the following 2-D minimization toy problem as an example.

2-D Example

Consider the minimization of the following objective over 2 variables:

$$f(x, y) = (x^2 + y^2 - 1)^2 - 0.5x \tag{4.2}$$

A contour plot of the objective is shown in the left of Fig. 4.12.

We can see intuitively that the objective 4.2 has two parts: $(x^2+y^2-1)^2$ which quantifies a distance away from the circumference of the unit circle, and $-0.5x$ which adds a linear tilt to the objective. By symmetry in y, the global minimum must lie on the x-axis, with $y^* = 0$. Due to the linear tilt, we can also roughly guess that the x-coordinate of the global minimum will be positive and very close to the unit circle, so $x^* \approx 1$. The linear tilt also makes this the *only* global minimum. Additionally, there are no *local* minima

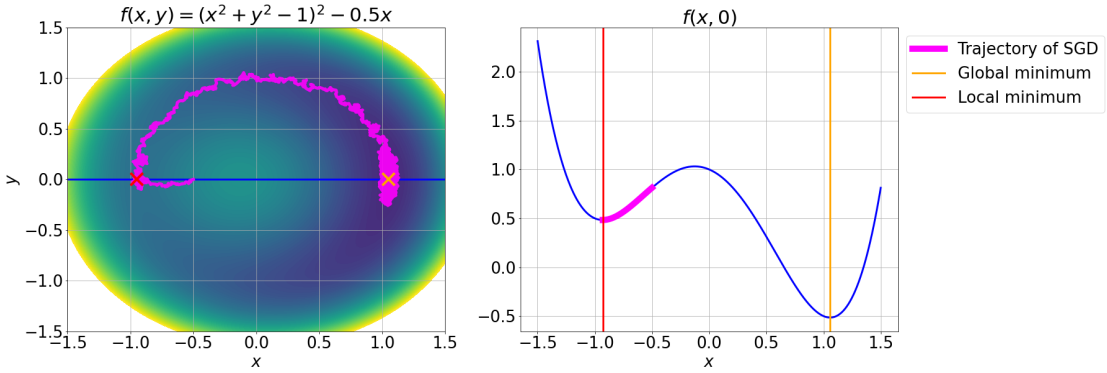


Figure 4.12: Visualizing the 2-D toy problem objective. Restricting the function to the line $y = 0$ which contains the global minimum introduces a local minimum at $x \approx -1$ which was originally a saddle point in the 2-D optimization problem. Notice that SGD initialized from $(-0.5, 0)$ can escape the saddle in the 2-D case, but gets stuck in the local minimum in the 1-D case.

as well. It is easy to see therefore that gradient descent will find this global minimum from any initialization (except for the local maximum at (0, 0)).

Now, if we follow the approach of parameter control and attempt to fix y to y^*, and only optimize over a single variable x we unexpectedly run into a problem. By restricting the function to the line $y = 0$, we have introduced a 1-D local minimum at $x \approx -1$ as seen in the right plot in Fig. 4.12. This means that gradient descent **will not** find the global minimum from all initializations, and will get stuck at the local minimum for a large range of possible initializations. This simple 2-D example shows that fixing parts of the network to their true parameter values and training the rest can actually hamper the optimization process.

4.4.4 Initializating g to g*

As we did with $\mathbf{w}$, we can attempt to just initialize the parameters of g_θ such that $g_\theta = g^*$ and train all parameters. However, the value of the true parameters for g_θ is not immediately clear like in the case of $\mathbf{w}$. We can approximate this scenario by initializing θ to those parameters obtained after training g_θ in a 1-D learning setting. Since we observed in section 4.1 that we can obtain validation losses very close to machine precision for 1-D learning, this approximation is justified. We can then present a hypothesis similar to hypothesis 4:

Hypothesis 6. *Learning the three-layer network with the parameters of g_θ initialized*

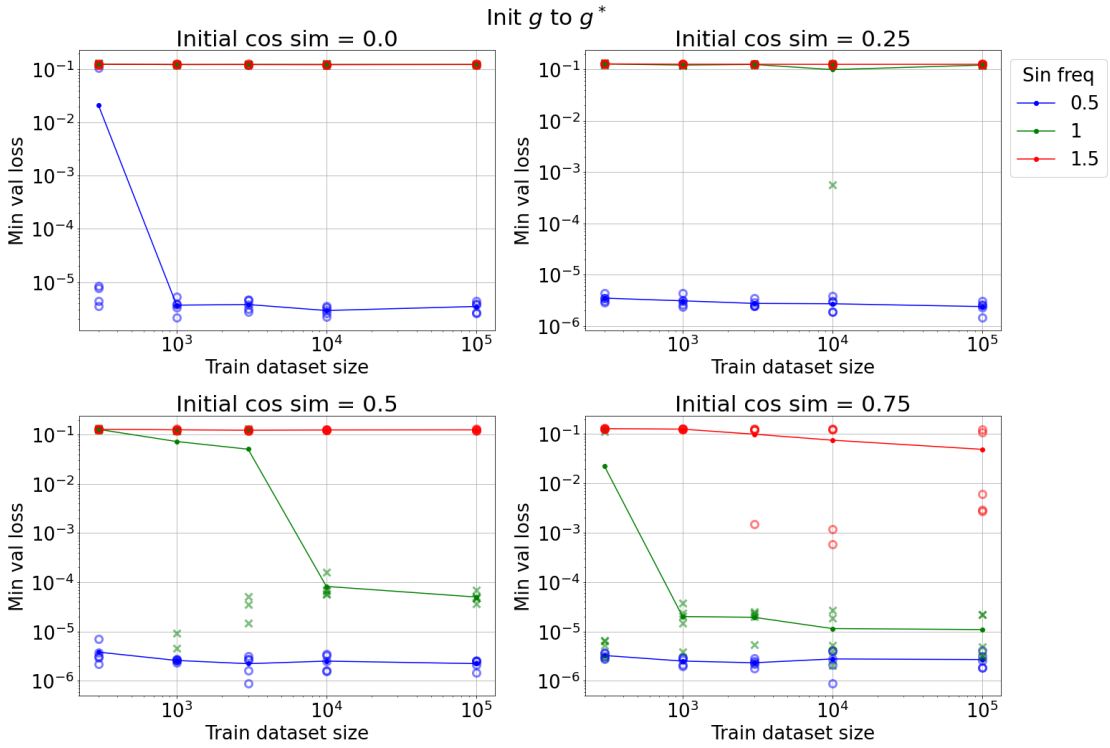


Figure 4.13: Initializing the parameters of g_θ with weights obtained using 1-D learning so that $g_\theta \approx g^*$. All parameters are then trained.

such that $g_\theta \approx g^*$ and training all parameters should have a higher success rate than random initialization.

The results of Fig. 4.13 suggest that hypothesis 6 is not clear. While initializing g_θ to g^* certainly does not hurt like it did when fixing g_θ to g^* in Fig. 4.11, it also does not seem to offer much benefit compared to the standard random initialization case in Fig. 4.5. One plausible explanation is again based on the observation made in section 4.3. If the 1-D learning component g_θ learns much faster than $\mathbf{w}$, then the initialization of g_θ would be quickly forgotten as it fits the low frequency component of the projected data using the random initial $\mathbf{w}$. However this is speculation, and the empirical results can neither confirm nor refute this claim.

4.5 Overparameterization

A primary focus of recent work in deep learning theory has been overparameterization [2]. Numerous results including the NTK and optimal transport based theories for convergence of gradient descent rely on overparameterization. The broad idea is that

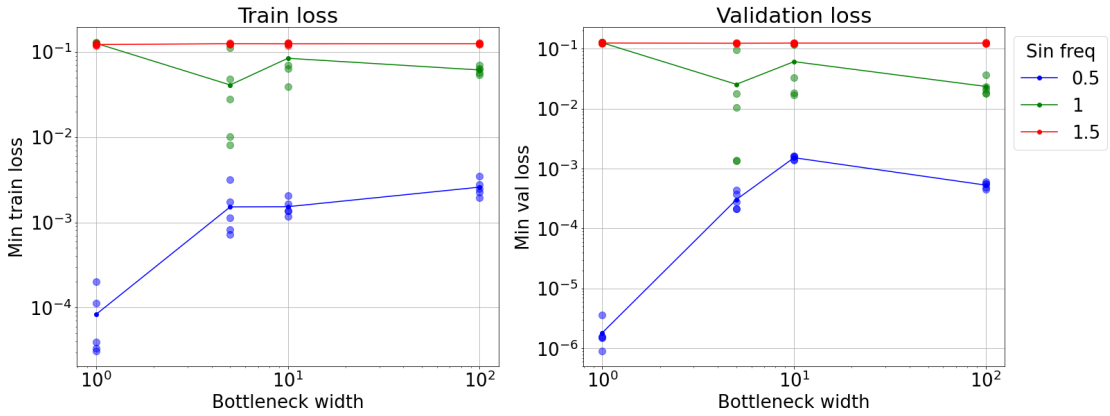


Figure 4.14: The effect of overparameterization. These experiments were run with a training dataset size of 10^5. Other dataset sizes were also tried, but the trend remained roughly the same so we exclude them to avoid clutter.

overparameterization may not have a negative impact in the deep learning setting as one would expect based on classical machine learning theory. This is based on the observation known as *double descent* [8], wherein generalization performance begins to improve again after the initial worsening as predicted by classical statistical learning theory. Following this direction, we study the effect of overparameterization in the setting of our case study, by increasing the 1-D bottleneck which is the output of the first linear layer. As a consequence of this, we lose out on a probing metric like the CosSim because g_θ is no longer a 1-D learner.

Hypothesis 7. *If the three layer network can learn with a bottleneck width of 1, increasing the bottleneck will improve the success rate.*

The results of Fig. 4.14 seem to refute hypothesis 7. Not only does overparameterization not help, it significantly worsens the performance for the case of 0.5 Hz. For the case of 1 Hz, it does seem to help to an extent. However, for 1.5 Hz, overparameterization has no significant impact on the success rate of learning. Note that this trend is seen even in the train loss, so it is not clear if overparameterization has a negative impact only on generalization or even on the optimization properties of the setting itself.

4.6 PL condition

A possible approach to developing a theoretical explanation for the observed learning dynamics in our case study is to follow the direction outlined in background sec-

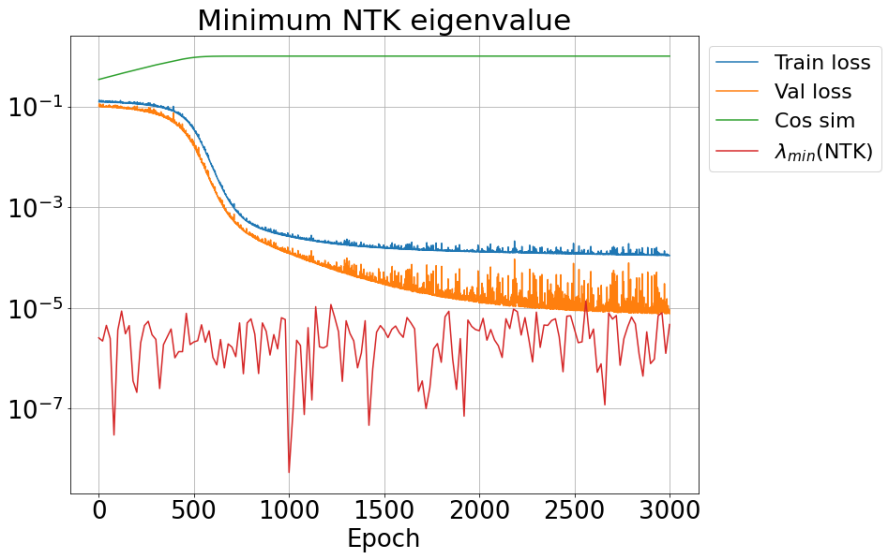


Figure 4.15: The minimum eigenvalue of the NTK does not provide much information in this particular problem, and the PL condition theory presented in [20] does not seem to explain the different phases of learning that we observe.

tion 2.2.2. A straightforward hypothesis to consider is:

Hypothesis 8. *Learning is explained by the PL condition theory described in background section 2.2.2. This means that the NTK is well conditioned as training progresses.*

Fig. 4.15 shows the metrics of a single run of the problem with all parameters trainable and randomly initialized, with an additional metric plotted – the minimum eigenvalue of the NTK evaluated on a minibatch of the training data. If the NTK is well-conditioned in a region around the global minimum of the optimization landscape, we expect the PL-condition to hold and gradient descent has linear convergence guarantees. However, the minimum eigenvalue of the NTK is very close to machine precision, and the NTK is certainly not well-conditioned. No significant change is observed even at the phase transition, so this direction cannot explain the different phases of learning that we observe. These empirical results seem to refute hypothesis 8.

4.7 Intriguing Optimization Phenomenon

An intriguing phenomenon that we stumbled upon in the process of running the numerous experiments presented above is related to the unexpected non-monotonic behavior of training performance with the size of the train dataset. In a general learning setting,

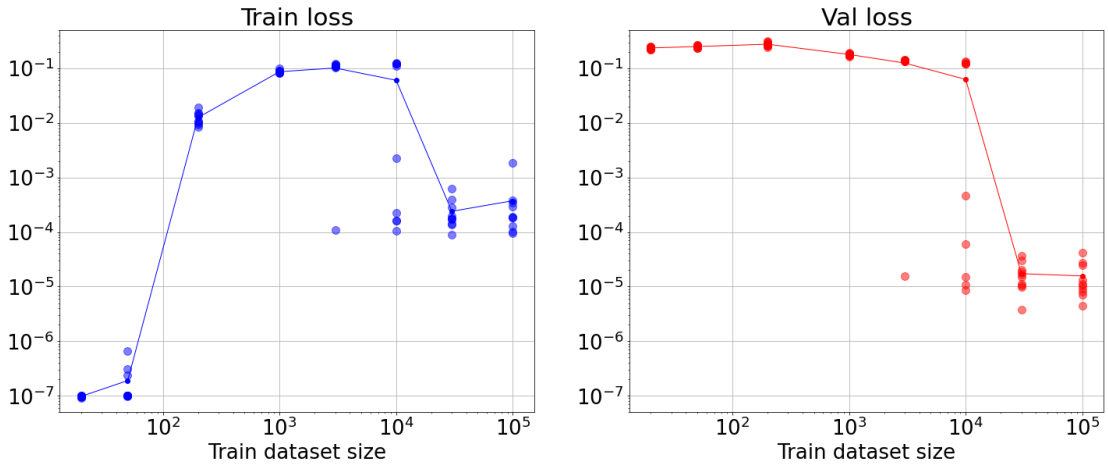


Figure 4.16: Unexpected non-monotonic behavior of train loss as the training dataset size is increased. These runs use an input dimension of $d = 100$.

basic intuition leads us to believe that fitting a larger training dataset with the same model is a harder problem. This arises simply from the fact that satisfying a larger number of equations with the same number of parameters has fewer solutions and is usually harder. However, in the context of our case study we do not observe this trend.

Fig. 4.16 shows the training losses and validation losses of multiple runs with the same hyperparameters except for train dataset size. All instances are run for the same number of gradient steps (not epochs as that would vary with the train dataset size) so that their results can be compared. We observe that the train loss hits machine precision for very small dataset sizes and correspondingly, the validation loss is very high indicating overfitting. The validation loss continues to monotonically decrease with increasing dataset size as is expected of learning problems. The training loss also initially rises with increasing dataset size as a larger dataset is harder to overfit with a fixed model. However, as the training dataset crosses a size of $\sim 10^4$ we observe a drop in the train loss. This is completely unexpected. The drop in the train loss is also accompanied by a sharp decrease in the validation loss. There is no immediate explanation for this intriguing phenomenon. Though we have presented validation loss results to provide the complete picture, note that this phenomenon is purely optimization related, as the unexpected trend is only in the train loss.

CHAPTER 5

Conclusion and Future Work

In this case study, we analyzed and empirically tested numerous hypothesis dealing with the learning dynamics of a particular three-layer deep network architecture. We motivated this architecture by combining existing architectures that have convincing theoretical explanations for their learning dynamics. Specifically, we extended the simple 1-D learner two-layer neural network to higher dimensions by projecting the input onto a 1-D line. This effectively amounts to a cascade of two linear layers followed by a non-linearity. Although a composition of linear layers is equivalent to a linear function, the learning dynamics are highly non-convex and non-trivial. There is significant theory analyzing the inductive bias of gradient descent on deep linear networks. As such, the architecture we studied is the simplest three-layer architecture with non-trivial learning dynamics that does not have a convincing theoretical explanation yet. A further advantage of studying this architecture is the ability to probe intermediate layer outputs and derive useful metrics like the CosSim to glean further insight into the learning process. This allowed us to formulate and test numerous hypothesis which would not be easily testable in standard deep learning settings.

Through the numerous experiments conducted, we discovered intriguing and counter-intuitive qualitative phenomenon that serve as clues towards developing a formal theory for learning. Specifically, the identification of phases in the learning process, combined with the negative impact of fixing g_θ to g^* in the optimization process strongly suggest that saddle point dynamics are coming into play. This presents a promising direction for future theoretical work in this setting. Additionally, the ability to separately experiment with the 1-D learner and $\mathbf{w}$ components during learning indicate that theoretical explanations could also use this feature in their analysis. The inability of existing theories to sufficiently explain the observations suggest that something richer and more complex is needed.

REFERENCES

[1] **Allen-Zhu, Z.**, **Y. Li**, and **Y. Liang** (2018). Learning and generalization in overparameterized neural networks, going beyond two layers. *arXiv preprint arXiv:1811.04918*.

[2] **Allen-Zhu, Z.**, **Y. Li**, and **Z. Song** (2018). A convergence theory for deep learning via over-parameterization. *arXiv preprint arXiv:1811.03962*.

[3] **Arora, S.**, **N. Cohen**, **N. Golowich**, and **W. Hu** (2018). A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*.

[4] **Arora, S.**, **N. Cohen**, **W. Hu**, and **Y. Luo**, Implicit regularization in deep matrix factorization. *In Advances in Neural Information Processing Systems*. 2019.

[5] **Arora, S.**, **S. S. Du**, **W. Hu**, **Z. Li**, **R. R. Salakhutdinov**, and **R. Wang**, On exact computation with an infinitely wide neural net. *In Advances in Neural Information Processing Systems*. 2019.

[6] **Arora, S.**, **S. S. Du**, **W. Hu**, **Z. Li**, and **R. Wang** (2019). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*.

[7] **Bach, F.** (2020). Gradient descent for wide two-layer neural networks: Global convergence. link. *Machine Learning Research Blog, Francis Bach*.

[8] **Belkin, M.**, **D. Hsu**, **S. Ma**, and **S. Mandal** (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, **116**(32), 15849–15854.

[9] **Chizat, L.** and **F. Bach**, On the global convergence of gradient descent for over-parameterized models using optimal transport. *In Advances in neural information processing systems*. 2018.

[10] **Chizat, L.**, **E. Oyallon**, and **F. Bach**, On lazy training in differentiable programming. *In Advances in Neural Information Processing Systems*. 2019.

[11] **Daneshmand, H.**, **J. Kohler**, **A. Lucchi**, and **T. Hofmann** (2018). Escaping saddles with stochastic gradients. *arXiv preprint arXiv:1803.05999*.

[12] **Dauphin, Y. N.**, **R. Pascanu**, **C. Gulcehre**, **K. Cho**, **S. Ganguli**, and **Y. Bengio**, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *In Advances in neural information processing systems*. 2014.

[13] **Du, S. S.**, **J. D. Lee**, **H. Li**, **L. Wang**, and **X. Zhai** (2018). Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*.

[14] **Du, S. S.**, **X. Zhai**, **B. Poczos**, and **A. Singh** (2018). Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*.

[15] **Dwaraknath, R. V.** (2019). Understanding the neural tangent kernel. link.

[16] **Gunasekar, S.**, **J. D. Lee**, **D. Soudry**, and **N. Srebro**, Implicit bias of gradient descent on linear convolutional networks. *In Advances in Neural Information Processing Systems*. 2018.

[17] **Jacot, A.**, **F. Gabriel**, and **C. Hongler**, Neural tangent kernel: Convergence and generalization in neural networks. *In Advances in neural information processing systems*. 2018.

[18] **Lewkowycz, A.**, **Y. Bahri**, **E. Dyer**, **J. Sohl-Dickstein**, and **G. Gur-Ari** (2020). The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*.

[19] **Li, Y.** and **Y. Liang**, Learning overparameterized neural networks via stochastic gradient descent on structured data. *In Advances in Neural Information Processing Systems*. 2018.

[20] **Liu, C.**, **L. Zhu**, and **M. Belkin** (2020). Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv preprint arXiv:2003.00307*.

[21] **Nakkiran, P.**, **G. Kaplun**, **D. Kalimeris**, **T. Yang**, **B. L. Edelman**, **F. Zhang**, and **B. Barak** (2019). Sgd on neural networks learns functions of increasing complexity. *arXiv preprint arXiv:1905.11604*.

[22] **Saxe, A. M.**, **J. L. McClelland**, and **S. Ganguli** (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

[23] **Shwartz-Ziv, R.** and **N. Tishby** (2017). Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

[24] **Szegedy, C.**, **W. Liu**, **Y. Jia**, **P. Sermanet**, **S. Reed**, **D. Anguelov**, **D. Erhan**, **V. Vanhoucke**, and **A. Rabinovich**, Going deeper with convolutions. *In Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

[25] **Vaswani, A.**, **N. Shazeer**, **N. Parmar**, **J. Uszkoreit**, **L. Jones**, **A. N. Gomez**, **L. Kaiser**, and **I. Polosukhin**, Attention is all you need. *In Advances in neural information processing systems*. 2017.

[26] **Zhang, C.**, **S. Bengio**, **M. Hardt**, **B. Recht**, and **O. Vinyals** (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

[27] **Zou, D.**, **Y. Cao**, **D. Zhou**, and **Q. Gu** (2018). Stochastic gradient descent optimizes over-parameterized deep relu networks. *arXiv preprint arXiv:1811.08888*.