A Study of the Edge of the Stability in Deep Learning

by

Curtis Fox

B. Sc., University of British Columbia, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

August 2023

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

A Study of the Edge of the Stability in Deep Learning

submitted by **Curtis Fox** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Mark Schmidt, Associate Professor, Computer Science, UBC
Supervisor

Michiel van de Panne, Professor, Computer Science, UBC
Supervisory Committee Member

Abstract

Optimization of deep neural networks has been studied extensively, but our understanding of it is still very limited. In particular, it is still unclear how to optimally set hyperparameters such as the step size for gradient descent when training neural networks. We explore the issue of tuning the step size for full batch gradient descent, examining a proposed phenomenon in the literature called the "edge of stability". This refers to a phase of training for neural networks when the training loss non-monotonically decreases over time while the sharpness (the maximum eigenvalue of the Hessian matrix) oscillates around the threshold of precisely two divided by the step size.

In this thesis, we start by providing the necessary background to understand the current state of the art in this area. We then perform various experiments on the sharpness, and study its trajectory over the course of training with full batch gradient descent. Unlike the vast majority of the previous literature, we focus on investigating the sharpness with respect to the various layers of neural networks individually. We observe that different layers of a neural network have differing behavior in their sharpness trajectories. We focus on fully connected networks and examine how varying hyperparameters of the network, such as the number of layers or hidden units per layer, impact the sharpness. Finally, we explore how various architecture choices impact step size selection when training with gradient descent. We see that changing just one parameter of a deep neural network, such as the non-linear activation functions used in each layer, can greatly impact which step sizes can lead to divergence during training. This motivates further investigation into how architecture choices affect hyperparameter selection.

Lay Summary

Machine learning has far-reaching impacts in our world, and is applied in a variety of sectors and industries. However, our understanding of how more complicated models such as neural networks learn, is very much lacking. These models often require extensive tuning and this can be a very time-consuming process. Thus, considerable research has looked into how to tune these models faster. In this thesis, we explore a phase of neural network tuning called the "edge of stability". In this stage, the accuracy of the model becomes unstable and fluctuates, rather than decreasing on every step of the tuning process. However, in the long run the quality of the model predictions improve even with this instability. By gaining a better understanding of the "edge of stability," we hope to shed new light on how to effectively tune deep neural networks.

Preface

This master's thesis is original, unpublished, independent work of the author, Curtis Fox. In particular, the code used to produce the experimental results in this thesis has been written by Curtis Fox, and where code packages written by others were used in producing the experimental results, this has been explicitly cited in this thesis.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would first like to thank my supervisor Mark Schmidt for his tremendous support in not only pursing my research goals, but also for his personal guidance. Next I would like to thank my parents for their unwavering support in all my academic pursuits, whether that be during my undergraduate studies, my Master's degree, or for my upcoming PhD. Thank you to all of my labmates who assisted in my research but also made my time in the lab an enjoyable experience. I am particularly grateful for all the assistance that Frederik Kunstner gave me throughout my Master's degree. Finally. I would like to thank all my friends for their support over the years.

Dedication

In loving memory of my grandmother.

Chapter 1

Introduction

In the past decade, machine learning, and in particular deep learning, has become increasingly popular. It is now widely used throughout many sectors and fields. Many of the problems that arise in various machine learning applications such as computer vision or natural language tasks can be framed as a simple minimization problem. In particular, they can be written in the following form:

$$\min_{w} f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

This is referred to as a finite-sum minimization problem. In this setting:

- The number of training examples is n.
- The vector of parameters is w.
- Each $f_i(w)$ is the loss on data point i.
- The overall loss averaged over all data points is $f(w)$.
- The domain of w is $\mathbb{R}^d$.

There are many loss functions that one can choose from such as the cross-entropy or logistic loss functions. However, in this thesis we will focus on one of the most predominantly used loss functions, the Mean Squared Error (MSE) loss:

$$\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i(x_i, w) - y_i)^2$$

such that:

- The predicted label for data point x_i is $\hat{y}_i(x_i, w)$.
- The true label of data point x_i is y_i.

Algorithm 1 Gradient Descent

Input: stepsize η, initial iterate w_0

  for $i = 0, \ldots, T-1$ **do**

    $w_{i+1} = w_i - \eta \nabla f(w_i)$

  end for

Return: w_T

Figure 1.1: Gradient Descent

The goal of this type of optimization is to find a solution that minimizes the loss function. For some simpler models such as linear or logistic regression, under certain conditions there can be one unique global minimizer that satisfies the above problem. However, for many modern machine learning problems the underlying model may be non-convex, which means that stationary points of the loss function may not be global minima. In particular, this holds true for the deep neural network models, which is what we will focus on in this thesis. Since many local minima may exist, and we are not guaranteed to find a global minimum, we may have to be content with our algorithm of choice just finding a local minimizer. In order to find such a solution, we can make use of the gradient descent algorithm, which we will discuss in the next section. It is worth mentioning that for some neural network problems, gradient descent can find a global minimum if the network has a sufficiently large number of parameters [Zhang et al., 2017].

1.1 Gradient Descent

The gradient descent algorithm will be the basis for all the optimization discussed in this work. The main idea of the algorithm is to take steps in the direction of the negative gradient at the current iterate. The updates of this algorithm take the form $w_{i+1} = w_i - \eta \nabla f(w_i)$, where η is referred to as the step size (or learning rate) [Cauchy, 1847]. See Figure 1.1 for the pseudo code for constant step size gradient descent. For this variant of gradient descent, we use a fixed number of iterations. However, other variations exist which instead check some kind of termination condition, such as the gradient norm being sufficiently close to 0.

The difficulty in using the gradient descent algorithm comes from how to set the step size of the algorithm. There are many different step size schemes, the simplest being to use a constant step size. Others include step decay, polynomial step sizes, exponentially decreasing step sizes, the Barzilai-Borwein step size, and cosine annealing [Barzilai and Borwein, 1988; Lacoste-Julien et al., 2012; Li et al., 2021; Loshchilov and Hutter, 2017; Wang et al., 2021]. Depending on the particular problem at hand, different step size schemes can lead to faster convergence than others. However, it is worth mentioning that step size schedules that are justified theoretically may not always lead to the best results in practice. Finally, using too large of a step size can lead to the algorithm of choice diverging entirely. In order to theoretically justify the use of certain step sizes, various assumptions are made about the underlying structure of the model (or function)

of interest. The next section will go over one of the most commonly used assumptions in the analysis of optimization algorithms.

Before proceeding further, it is worth mentioning that another variant of gradient descent, called Stochastic Gradient Descent (SGD) is commonly used throughout machine learning [Robbins and Monro, 1951]. The key difference is that for SGD, only a subset of the training examples are used to compute the gradient on each step, leading to much faster iterations at the cost of progress per iteration. This is called mini-batching, and it is a useful technique often used for training large deep learning models [Devlin et al., 2019; Dosovitskiy et al., 2021; Vaswani et al., 2017]. Finally, in addition to standard gradient descent, adaptive step size variants also exist. These include Adam [Kingma and Ba, 2015], Adagrad [Duchi et al., 2011], and Adadelta [Zeiler, 2012]. In particular, Adam is often the algorithm of choice in practice when training large neural networks. However, these adaptive algorithms will not be the main focus of this thesis.

1.2 Smoothness

One of the most commonly used assumptions in the field of optimization is L-smoothness. We define it formally as follows.

Definition: The function f is L-smooth if there exists a constant $L > 0$ such that:

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \quad \forall x, y \in \mathbb{R}^d$$

This definition can be thought of as requiring f to be upper bounded by a quadratic function. We call L the Lipschitz constant of ∇f. This definition gives us a way to upper bound the growth of functions we analyze. Equipped with this definition, we can give the rate of convergence for gradient descent under the following assumptions:

1. The step size $\eta = \frac{1}{L}$
2. f is L-smooth
3. f is bounded below by f^*

We take the following result from Nesterov [2018]:

Theorem - Rate of Convergence of Gradient Descent: Under the given assumptions, for some constant $c \geq 0$ and a function f, the following holds after $T+1$ steps of the form $w_{i+1} = w_i - \frac{1}{L}\nabla f(w_i)$:

$$\min_{i \in \{0,\ldots,T\}} \|\nabla f(w_i)\|^2 \leq \frac{cL}{T+1}(f(w_0) - f^*)$$

We consider a step size of $\frac{1}{L}$ since this is the optimal step size under the given assumptions [Nesterov, 2018]. Additionally, when proving the rate of convergence for gradient descent on a smooth function, one

can show that if $\eta > \frac{2}{L}$, then algorithm isn't guaranteed to converge [Nesterov, 2018]. However, even though the L-smooth assumption is appropriate for simpler models such as linear or logistic regression, as discussed in the recent work by Cohen et al. [2021], this assumption may not be well justified or appropriate for training deep neural networks. Before we discuss the main idea of this thesis, we'll briefly review neural networks as they will be key to our discussion.

1.3 Neural Networks

As mentioned earlier, neural networks have become highly prevalent in many applications. The following definition gives a formal definition of the simplest type of neural network model, the fully connected neural network (also known as the Multi-layer Perceptron (MLP)):

Definition: An MLP with 3 layers takes the following form:

$$f(X, W_1, b_1, W_2, b_2, W_3, b_3) = W_3(h_2(W_2 h_1(XW_1 + b_1) + b_2)) + b_3$$

where:

- The input matrix is X.
- The weight matrix and bias vector of the input layer are W_1 and b_1.
- The weight matrix and bias vector of the hidden layer are W_2 and b_2.
- The weight matrix and bias vector of the output layer are W_3 and b_3.
- Each h_i is the non-linear activation function of layer i.

It is standard to refer to the weights and bias variables as the network parameters. Note that we can easily extend this definition to define deeper networks with more layers by composing further weight matrices, bias vectors, and activation functions. Each additional weight matrix that we add constitutes an additional hidden layer. Another commonly used term when discussing neural networks is the width (often referred to as the number of hidden units per layer, which is the term we will use in this work). This can vary between layers, and using a different number of hidden units will lead to different sized weight matrices and bias vectors. Finally, sometimes for theoretical purposes the non-linear activation functions are left out, and the resulting model is called a linear neural network. However, this is generally not used in practice because it degenerates to a linear model. Using the finite-sum minimization framework discussed earlier, we want to find parameters that minimize the function of interest. However, in practice it is common to find only an approximate solution, rather than converging to an exact minimizer. Two of the main reasons for this is faster convergence speed, or for better generalization error. By generalization error, we mean the error the model incurs on data not seen in the model training process. This leads one to wonder how we should update the parameters so that we can find a local minimizer. This involves the use of an optimizer such as

gradient descent, which iteratively updates the network parameters. This process of updating the parameters of a neural network is referred to as backpropagation [Linnainmaa, 1970; Rosenblatt, 1961; Rumelhart et al., 1986]. However, as discussed earlier, appropriately choosing a constant stepsize or stepsize schedule for gradient descent can be very challenging, and this question motivates both the work in this thesis and extensive deep learning research.

Another question of interest that we will only briefly discuss is how to initialize the parameters of a neural network. This is by no means a simple issue. There are a variety of ways to initialize neural network parameters, with some of the commonly used techniques being the Xavier and Kaiming initializations [Glorot and Bengio, 2010; He et al., 2015]. Some work has explored how to initialize neural network parameters [Kumar, 2017; Narkhede et al., 2022], but this line of research is still very much open.

1.4 Introducing the Edge of Stability

We previously alluded to how classical smoothness assumptions on function growth may not hold for neural networks. To understand why this is, we present a phenomenon called the "edge of stability," which was formally introduced in Cohen et al. [2021]. Before we can discuss this idea further, we need to introduce the following definition:

Definition: The sharpness of a neural network, which we denote as λ, is the maximum eigenvalue of the Hessian of the loss function with respect to all the parameters of the neural network.

Intuitively, the sharpness can be thought of as the maximum rate of change of the gradient. As observed in Cohen et al. [2021], this quantity, along with the training loss, have very interesting behavior over the course of training across a number of architectures and datasets. Assuming we use a constant step size satisfying $\eta > 0$, we see the following phases empirically during neural network training:

1. In the initial stages of training, the training loss (mostly) monotonically decreases and the sharpness (mostly) monotonically increases. This phase is referred to as progressive sharpening.

2. Once the sharpness reaches or exceeds the threshold of $\frac{2}{\eta}$, the sharpness begins to oscillate around the value $\frac{2}{\eta}$, and no longer monotonically increases. The training loss continues to decrease over time, but does so non-monotonically. This phase is referred to as the edge of stability (EOS). Gradient descent tends to remain in this phase once it is reached unless the step size changes.

This is of interest because it is in direct contrast with theory from convex optimization. In this setting, if $\lambda > \frac{2}{\eta}$ then the gradient descent steps taken will be too large and gradient descent is not guaranteed to converge. Note that for L-smooth twice differentiable functions, $\lambda \leq L$, and $L > \frac{2}{\eta}$ is a condition under which gradient descent may diverge (which we discussed earlier in Section 1.2). However, this restriction on the step size does not seem necessary in neural network training. As we see in Figure 1.2, for each step size η used, the EOS is reached at roughly the threshold $\frac{2}{\eta}$. Notably, for larger stepsizes, the EOS is

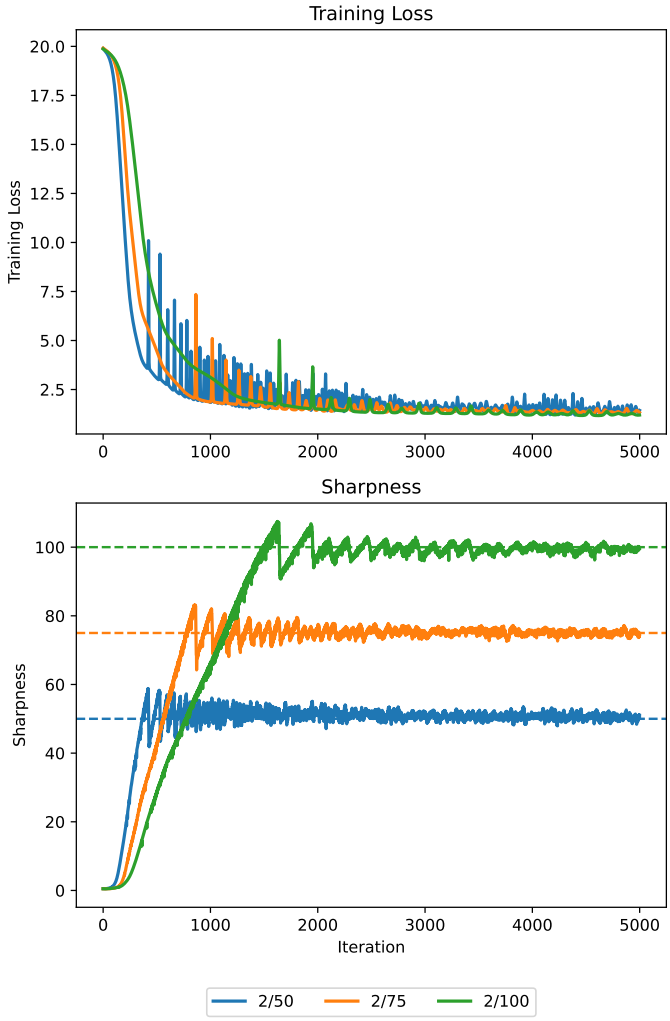


Figure 1.2: Edge of Stability with 3 different step sizes. The model trained is a 3-layer fully connected neural network with Tanh activations, using synthetic regression data.

reached more quickly than for smaller step sizes. Once the above threshold is reached, the loss sometimes dramatically increases initially, but still decreases over time. One of the key takeaways from this experiment and the work by Cohen et al. [2021] is that it may be appropriate to use large constant step sizes that lead to the EOS being reached during training of neural networks. However, at this point in time it is not entirely clear what role the EOS plays in the convergence speed of gradient descent. In the next chapter we will examine the literature related to the EOS and attempt to gain a better understanding of it.

1.5 Summary of Contributions

We briefly outline the contributions of this thesis as follows:

1. In Chapter 2, we discuss relevant literature around the edge of stability.

2. In Chapter 3, we give experimental results analyzing layer wise sharpness. In particular, how the sharpness trajectories vary between different layers of a neural network. We will then give experiments comparing layer wise step sizes schemes to global step sizes, comparing their sharpness and training loss trajectories.

3. In Chapter 4, we give experimental results examining how the range of step sizes that lead to divergence for gradient descent and SGD (within some number of training iterations) changes depending on neural network architecture choices.

4. Finally in Chapter 5 we review insights gained from these experiments and discuss future research directions.

Chapter 2

Background

In this chapter, we introduce relevant background and related work on the "edge of stability."

2.1 Understanding the Edge of Stability

Even though the work by Cohen et al. [2021] shows that the EOS occurs in a variety of settings, they do not provide much justification for why the phenomenon occurs. This has led to a flurry of papers attempting to explain and understand the EOS from a variety perspectives, some of which we will discuss in this section. The paper by Ahn et al. [2022b] is one of the first attempts to characterize and better explain the EOS phenomenon. They theorize that gradient descent enters the EOS due to a lack of flat minima near the gradient descent trajectory. The work by Ma et al. [2022] explains the EOS using a property of subquadratic growth (a loss function which grows more slowly than a quadratic as you move further away from the minimizer) near minimizers when training neural networks. The work by Arora et al. [2022] takes a different perspective, and attempts to understand the EOS by focusing on two particular settings. They prove the EOS occurs, under some assumptions on the function f, here in the following two settings:

1. The first setting is what they call normalized gradient descent, which is gradient descent with step size $\eta_i = \frac{\eta}{\|\nabla f(w_i)\|}$ on each iteration i.
2. The other setting is considering gradient descent on loss functions of the form $\sqrt{f}$, where $f^* = 0$. For clarity, they take a loss function f satisfying certain regularity conditions and take the square root of this function. They then consider full batch gradient descent on this modified loss function.

Finally, deep neural networks aren't the only setting where the EOS is observed. In the work by Agarwala et al. [2022], they consider what they call second-order regression models, which are models where there is a quadratic dependence on the parameters. This corresponds to a quartic loss function, as opposed to the usual squared loss function seen in machine learning. In particular, they observe that even in this setting, the EOS can be observed. Even though these works provide some insight into intricacies of the EOS, gaining a better understanding of the phenomenon may require focusing on simpler settings where the EOS occurs. This is exactly what we will discuss in the next section.

2.2 The Edge of Stability in a Simpler Setting

To better understand the EOS on simpler models, the work by Zhu et al. [2023] focuses on the case of a 4 layer scalar linear network, where each layer has only one hidden unit. The loss function takes the following form:

$$f(w_1, w_2, w_3, w_4) = \frac{1}{2}(1 - w_1 w_2 w_3 w_4)^2$$

where each w_i is a scalar. The authors make further simplifications to make the analysis easier, and give various theoretical results. They give both formal and informal results under this model. We present one such informal result here for brevity, since the formal results are very detailed. From their paper:

Theorem - Sharpness Concentration: For any learning rate η less than some constant, there is a constant size region B_η such that the trajectory of gradient descent with step size η from all initializations in B_η converge to a global minimum with sharpness in $(\frac{2}{\eta} - \frac{20}{3}\eta, \frac{2}{\eta})$

Essentially this says that as long as the step size picked is not too large, with appropriate initialization one can quantify the sharpness of the resulting minimum that gradient descent converges to. Even though the setting discussed in this work is unrealistic and unlikely to come up in practice, this paper takes a step towards better understanding the properties of the solutions found by gradient descent in the EOS regime. The key takeaway from work studying a simplistic model is that the EOS phenomenon can still occur, and thus is not necessarily a product of the more complicated models seen in practice that make use of many layers and non-linear activation functions. Understanding precisely what model properties induce the EOS, and under what additional settings it can occur, will make for interesting future research directions.

2.3 Justifying the Edge of Stability using Third Order Derivatives

Some recent work attempts to provide justification and analysis of the EOS by considering the third order derivatives with respect to model parameters. As stated earlier, for a quadratic function, selecting a step size of $\eta > \frac{2}{L}$ causes gradient descent to diverge. In the work by Chen and Bruna [2022], they give a simple setting where selecting a step size slightly larger than this threshold doesn't lead to immediate divergence of gradient descent. The first theorem of their paper considers scalar functions, and makes the following assumptions, where $f^{(3)}$ and $f^{(4)}$ refer to the 3rd and 4th derivatives respectively:

1. The first 3 derivatives of f exist and are continuous around a local minimum w^*
2. $f^{(3)}(w^*)$ is non-zero
3. $\frac{f^{(3)}(w^*)}{f^{(2)}(w^*)}$ is $O(1)$
4. $3(f^{(3)})^2 - f^{(2)}f^{(4)} > 0$ at $w = w^*$

5. The higher order derivatives are bounded as $O(1)$

Their theorem is as follows:

Theorem: Under the assumptions given above, for a starting point near w^* where $w_0 = w^* - \varepsilon$ such that $\varepsilon f^{(3)}(x^*) > 0$ and ε is sufficiently small, there exists a step size η for gradient descent starting from w_0 such that gradient descent will be back to w_0 in two steps and $\frac{2}{f^{(2)}(w^*)} < \eta < \frac{2}{f^{(2)}(w^*) - \varepsilon f^{(3)}(w^*)}$.

The main idea is that for a scalar function satisfying certain regularity conditions around a local minimizer, selecting a step size slightly larger than the $\frac{2}{L}$ threshold doesn't lead to immediate divergence of gradient descent, at least in a local neighborhood of this local minimizer. They note that this theorem isn't in contradiction with the divergence behavior for quadratic functions, since assumption 2 disqualifies quadratic functions.

The paper by Damian et al. [2023] has a more general analysis, where they consider a property they refer to as self-stabilization. This property says that as iterates move towards the minimizer, the cubic term in the Taylor expansion of the gradient biases optimization in the direction of the negative gradient of the sharpness, where the sharpness is considered as a function of the parameters. This bias decreases the sharpness, and this occurs until sharpness drops below $\frac{2}{\eta}$, leading to stabilization. This helps explain why gradient descent does not necessarily diverge for large step sizes in deep learning settings. This line of work considering the third order derivatives to explain the EOS is not yet well explored, and further insights could still be gained upon further exploration.

2.4 Sharpness, Choice of Stepsize, and Generalization Error

In addition to understanding how the EOS appears during the training process in deep learning, we will briefly discuss some related work exploring how sharpness, choice of step size, and generalization error are connected, both for full batch and stochastic gradient descent. In particular, using larger stepsizes when running gradient descent to train neural networks has been shown to find "flatter" minima [Lewkowycz et al., 2020; Wu et al., 2018]. In the work by Ahn et al. [2022a], they show that for a 2 layer RELU network model, large step sizes are required in order to learn what they refer to as "threshold neurons" (bias variables with negative values). As they discuss, these threshold neurons can be important for model generalization. In addition, extensive research has explored a possible connection between generalization error and the sharpness of the solution found by gradient descent. An algorithm called Sharpness-Aware Minimization (SAM) was developed with the main idea being to find flatter solutions and penalize sharper ones, and it has been shown to get improved generalization error on a variety of tasks compared to regular SGD [Andriushchenko and Flammarion, 2022; Foret et al., 2021]. Various works have since attempted to extend the standard SAM algorithm, with adaptive variants [Kwon et al., 2021], or by making it more efficient by requiring fewer gradient computations [Du et al., 2022; Liu et al., 2022]. The work by Ahn et al. [2023] proposes an algorithm called the "Randomly Smoothed Perturbation" algorithm which also searches

for flat minima by adding random perturbations to iterates and using the gradients at these points. However, even with extensive research that has been done, the relationship between the sharpness of minima and their associated generalization error is still not entirely clear. In particular, the answer may not be as simple as flatter solutions lead to better generalization, and a variety of factors, such as the batch size used for training, can play a role in generalization error [Even et al., 2023; Kaur et al., 2022].

Chapter 3

Examining the Edge of Stability

In this chapter we extend the experiments of Cohen et al. [2021]. While Cohen et al. [2021] analyze the overall sharpness behavior of the network, we plot the sharpness of each individual layer. Note that all experiments computing sharpness, both earlier in this paper and from this point on, make use of the PyHessian package [Yao et al., 2020] to compute the sharpness values. For this section, we use synthetic data for the experiments involving a layer-wise analysis of the sharpness. The reason behind this is to ensure that each layer of the neural network has the same number of hidden units, otherwise for many reasonable datasets we expect that the input layer will be significantly wider than all other layers. Thus, to ensure that the size of the input layer isn't a contributing factor to the sharpness behavior during training, we generate synthetic data so that the dimension of the input layer is the same as all other layers. For similar reasoning, we also ensure the output layer has the same number of hidden units as the rest of the layers. However, before proceeding to our experimental results, we outline our experimental settings in the next section, which we use for the experiments in this chapter as well as in the next chapter.

3.1 Experimental Settings

For all the experiments in this paper, we use the default parameters as given in Table 3.1, unless otherwise stated. As well, we make the following notes about our experiments:

- SGD with full batch is just normal gradient descent.
- Other parameters of the models not stated in the tables above, such as activation functions used, the number of hidden layers, and the number of hidden units per layer, are stated with each individual experiment as these are often varied.
- All hidden layers of any particular network use the same activation functions.
- All experiments computing the Hessian of a network, and the corresponding sharpness require the use of the power method (this is how the maximum eigenvalue of the Hessian is computed, and note that the value computed is an approximation to the true eigenvalue). In all cases, the maximum eigenvalue

Table 3.1: Experimental Details

Optimizer	SGD
Batch Size	Full Batch
Training Examples	5000
Weight Initialization	Pytorch Default
Bias Initialization	Pytorch Default
Loss Function	MSE
Bias Variables	In all layers
Optimizer Iterations	5000

Table 3.2: Dataset Details

Dataset	Number of Training Examples	Data Dimensions	Reference
MNIST	60,000	28 by 28 by 3	[LeCun et al., 1998]
CIFAR-10	60,000	32 by 32 by 3	[Krizhevsky et al., 2009]

of the Hessian is computed using the *eigenvalues* function in the PyHessian package [Yao et al., 2020], with the *maxIter* parameter set to 100 and the *top_n* parameter set to 1. For the *tol* parameter, the default setting is used.

3.1.1 Datasets

In this section we outline the details of our synthetic data generation, as well as the real datasets we used.

Synthetic Regression Data

We generate a synthetic regression dataset with n training examples and d dimensions as follows:

1. Generate the entries of an n by d matrix X and a d by d matrix V from the standard normal distribution
2. Compute an n by d matrix $Y = XV^T$
3. Add noise to the matrix Y by subtracting a matrix of the same dimensions with entries generated from the standard normal distribution

This generates a dataset X with labels Y (where each row of Y_i corresponds to a potentially multi-dimensional label for example X_i).

Real Datasets

In Table 3.2 we give the details for the real datasets that we use. Note that we only use a subset of the training examples in our experiments, usually set either 5000 in Chapter 3 and 100 in Chapter 4.

3.2 Sharpness by layer

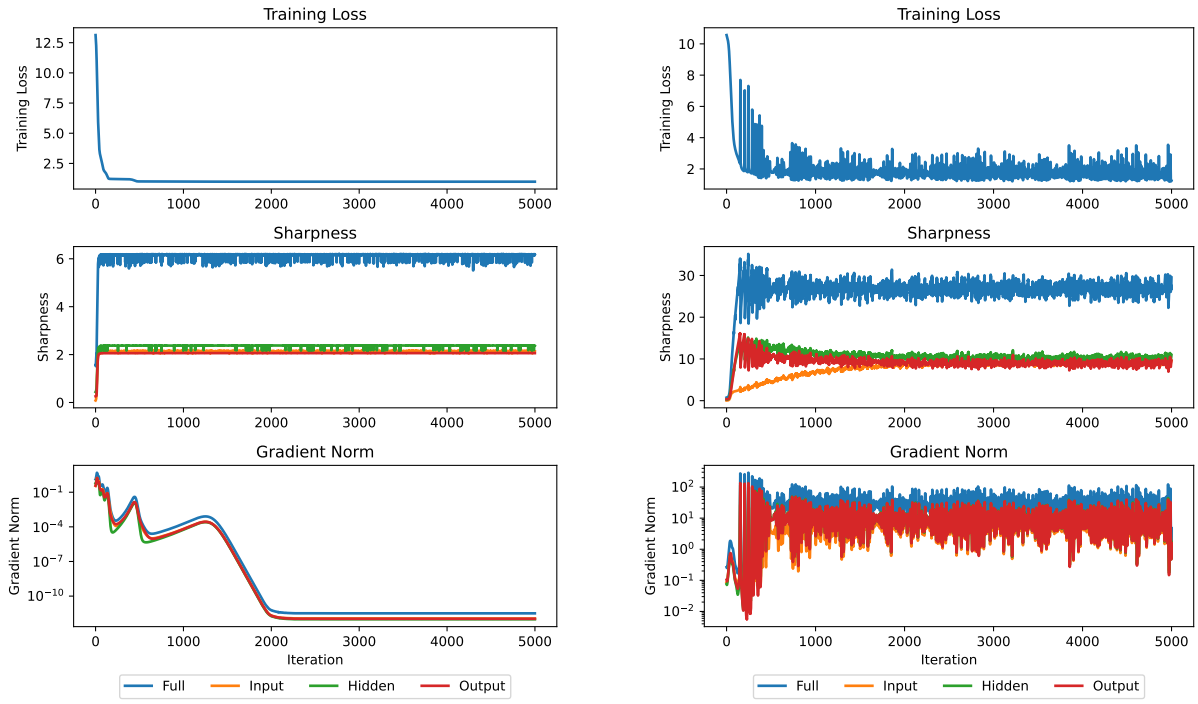
In this section we examine how constant step sizes impact the sharpness across different layers. This is something that is not well explored in the literature, and understanding how the sharpness of different layers changes over time when training, could potentially give further insights into how to speed up neural network training. In particular, this could give insight into selecting different step sizes for different layers. To ensure it's clear what we mean by the sharpness of an individual layer, we introduce the following definition:

Definition: The sharpness of layer i of a neural network, which we denote as λ_i, is the maximum eigenvalue of the Hessian with respect to the parameters W_i and b_i while holding the remaining parameters of the network constant.

However, one complication in analyzing how the sharpness changes over time across layers is that even just changing one aspect of the model architecture can lead to wildly different behavior in not only the sharpness of the overall network (illustrated in Cohen et al. [2021]), but also in the sharpness of individual layers. In Figures 3.1 and 3.2, we generate synthetic data from the same distribution for each experiment, only changing which activation function we use for the model. In Figures 3.3 and 3.4, we perform similar experiments but instead we vary the network width. Finally, in Figures 3.5 and 3.6 we compare across networks of different depths. Note that in each legend, "Full" refers to the sharpness with respect to all the weight and bias parameters of the network, rather than any one particular layer. We choose to use a step size of $\frac{2}{25}$ for these experiments as it is large enough for us to see the EOS in relatively few training iterations while small enough to ensure gradient descent doesn't diverge. For each of these experiments, we only do one run. This is because they are all done using full batch gradient descent. Thus, we don't expect to see the large variation between different runs that one might see when using smaller batches for training. For the purposes of this thesis, our focus is specifically on how the EOS impacts training. Thus, we do not plot the validation loss, and leave the discussion of generalization for future work.

From these plots, one key observation that we make is that not only does sharpness of the overall network oscillate, but it seems as though the sharpness of each individual layer oscillates around its own threshold value as well. In Figures 3.1 and 3.2 we see that changing only the activation functions of the neural network leads to different behavior of the sharpness of each individual layer. In particular:

- For the linear and ReLU networks, the input, hidden, and output layers have comparable behavior in the growth of the sharpness over time.
- For the sigmoid activation, the input and hidden layers see much more growth in their sharpness over time than the output layer does.
- For the tanh activation, all 3 layers show noticeably different behavior in sharpness growth during training.



(a) Linear **(b)** ReLU

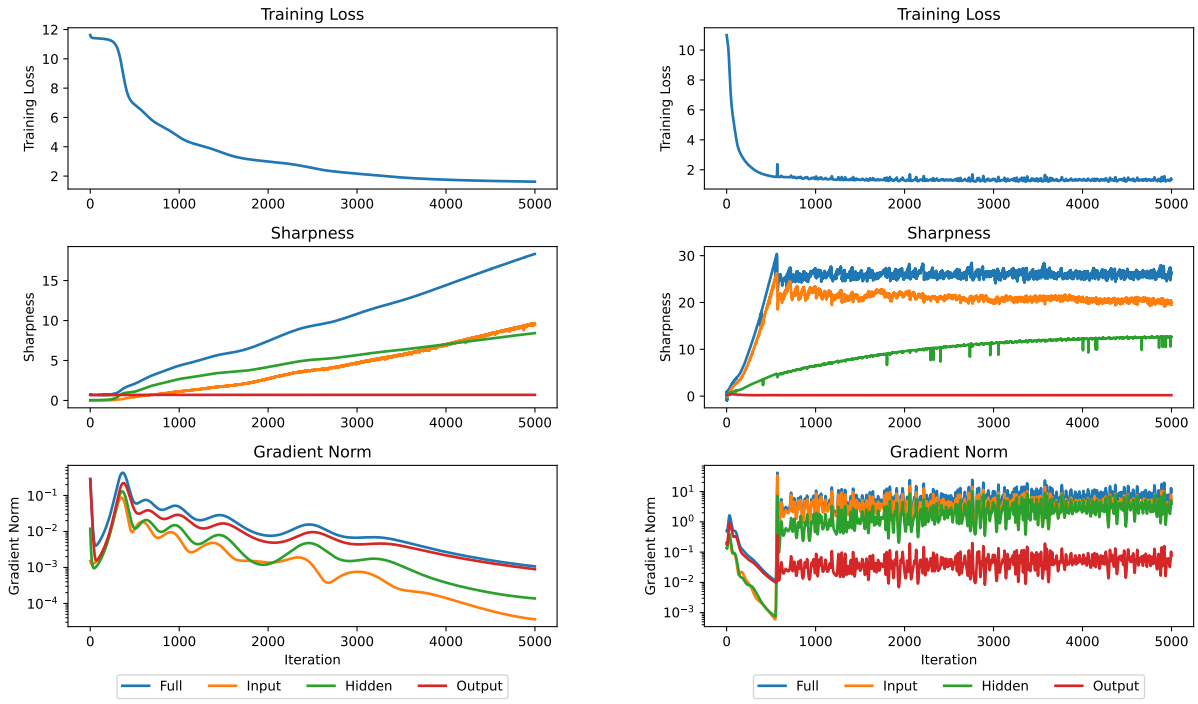
Figure 3.1: Edge of Stability compared across different activation functions using a common stepsize of $\frac{2}{25}$. Each network has 3 layers and 10 units in each layer. The data used is synthetic regression data.

When we instead vary the width and depth of the networks in Figures 3.3 through 3.6, we observe the following:

- When varying the number of hidden units, we see that the sharpness behavior per layer doesn't change much across different network widths.
- When instead changing the network depth, the main trend that we observe is that as one increases the network depth, the growth of the sharpness of each layer, as well as their points of oscillation, seem to become more similar.

One key takeaway from all these experiments is that even just changing the activation functions used in each layer can affect both the sharpness behavior with respect to the whole network and with respect to each layer. In addition, it's not the case that particular layers, such as the input layer, will always go through more progressive sharpening than other layers. Thus, the behaviour of any particular layer may not easily be predicted from the architecture, and further research may be required to better predict expected sharpness behaviour. See the Appendix for further layer-wise experiments, where we use a different step size than $\frac{2}{25}$.

The point of doing such comparisons is to illustrate how various properties of neural network architectures can impact not only sharpness behavior, but what possible implications this has for neural network



(a) Sigmoid **(b)** Tanh

Figure 3.2: Edge of Stability compared for further activation functions using a common stepsize of $\frac{2}{25}$. Each network has 3 layers and 10 units in each layer. The data used is synthetic regression data.

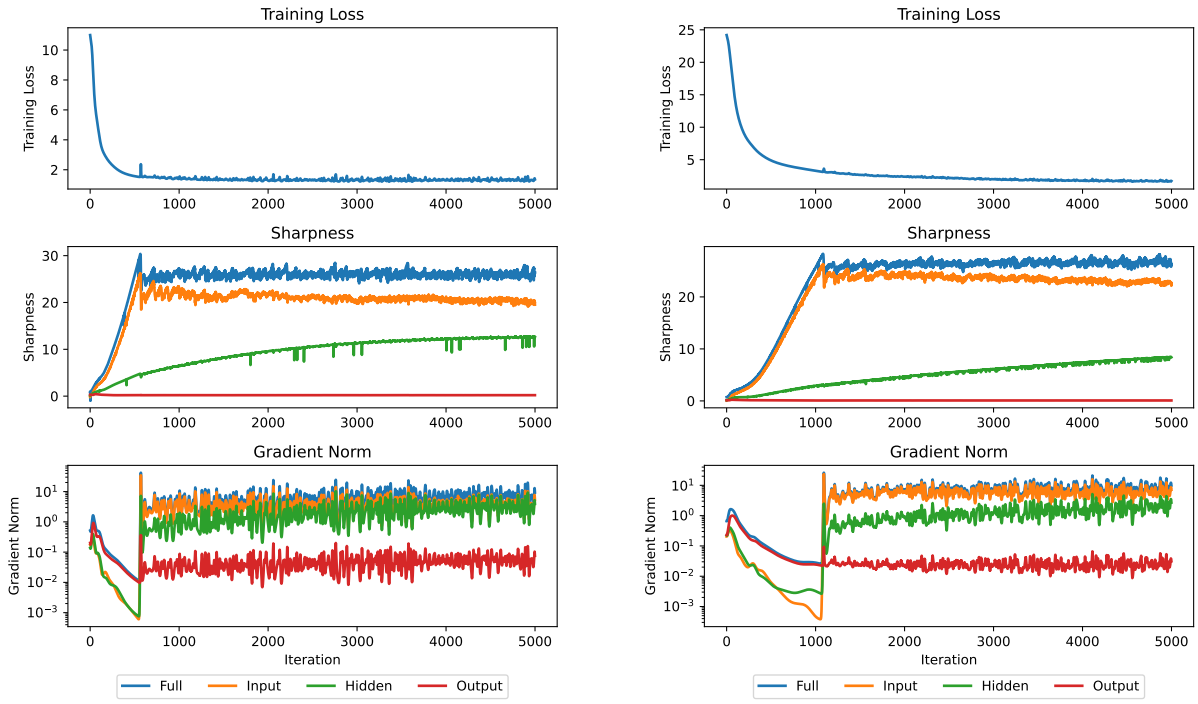
training. This may be particularly true if one wants to use different step sizes for different network layers. Also note that in addition to plotting the loss and sharpness, we also keep track of the gradient norms with respect to the network parameters. In plotting the gradient norms in our experiments, we made the following observations:

- In the early stages of training, or during the progressive sharpening phase, the gradient norms tend to decrease over time (although often non-monotonically).
- Once the EOS is reached, the gradient norms of both the overall network, as well as the individual layers, rapidly increase before stabilizing around some threshold.

Similarly to the sharpness, we see that the gradient norm oscillates around some threshold value once the EOS is reached. In particular, it seems as though the behavior of the sharpness and the gradient norms are similar once the EOS is reached. This leads to the following question:

Question: Is there a way to characterize the thresholds that the gradient norms of both the full network and individual layers will oscillate around (like the $\frac{2}{\eta}$ threshold for the sharpness of the full network)?

Some work has characterized the behaviour of the gradient norm for gradient descent in terms of the step



(a) 10 Hidden Units

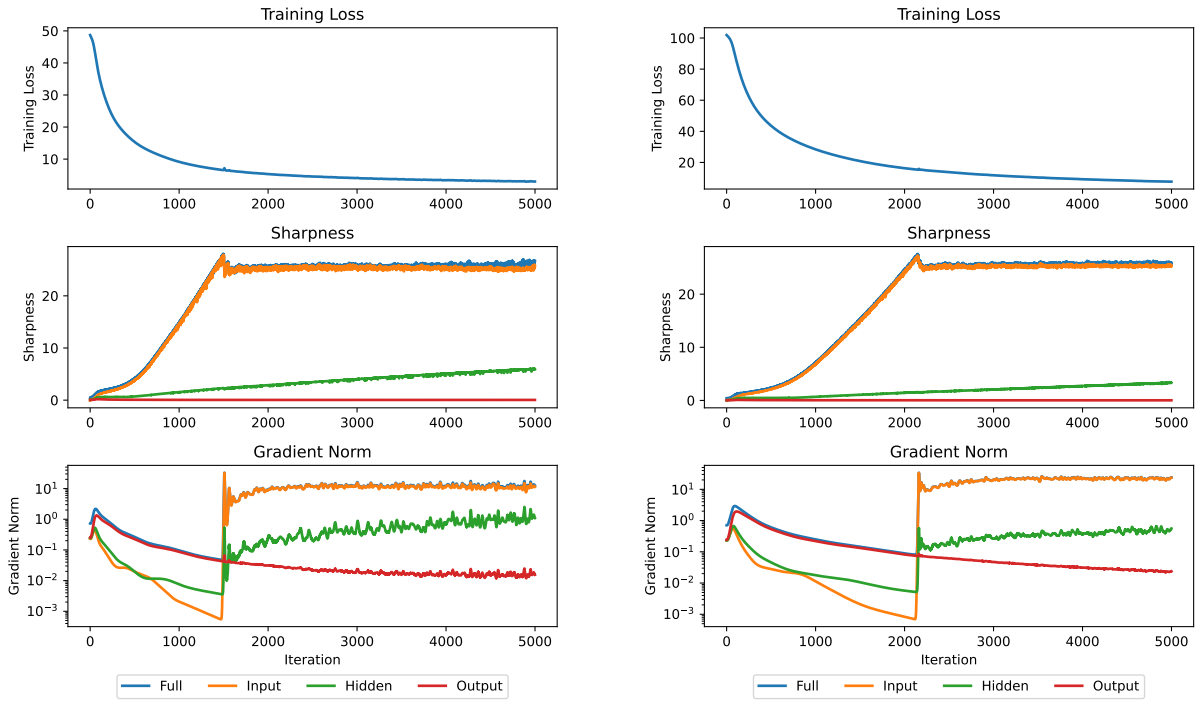
(b) 25 Hidden Units

Figure 3.3: Edge of Stability compared across different network widths with tanh activation functions using a common stepsize of $\frac{2}{25}$. Each network has 3 layers, and all layers have the same number of hidden units within each trained network. The data used is synthetic regression data.

size, variance of the stochastic gradients, and the batch size [Friedlander and Schmidt, 2013; Ghadimi and Lan, 2013]. However, just like the sharpness of the individual layers, characterizing the gradient norms of the individual layers is still an open question that warrants further exploration. Before moving onto the next section, we make one last point about how sharpness and the individual layers of a network are related. In particular, in the work by Wang et al. [2022], they perform experiments (see the appendix of their paper) showing that freezing layers during training leads to the sharpness of the overall network increasing more slowly. Notably, the more layers that are frozen, the slower progressive sharpening occurs. However, unlike us, they do not look at the sharpness of any particular layers and only the sharpness with respect to the entire network.

3.3 Setting step sizes by layer sharpness

As discussed in earlier work, setting the step size of gradient descent as $\frac{1}{\lambda_t}$, where λ_t is the sharpness on iteration t of gradient descent, is a poor heuristic [Cohen et al., 2021]. The motivation for using this step size scheme comes from convex optimization, where a step size of $\frac{1}{L}$ is optimal for a smooth quadratic (where $\lambda \leq L$ in this setting). However, as Cohen et al. [2021] shows, appropriately choosing a large constant step size leads to far faster convergence than using this scheme. Similar to their results, we see from Figure 3.7



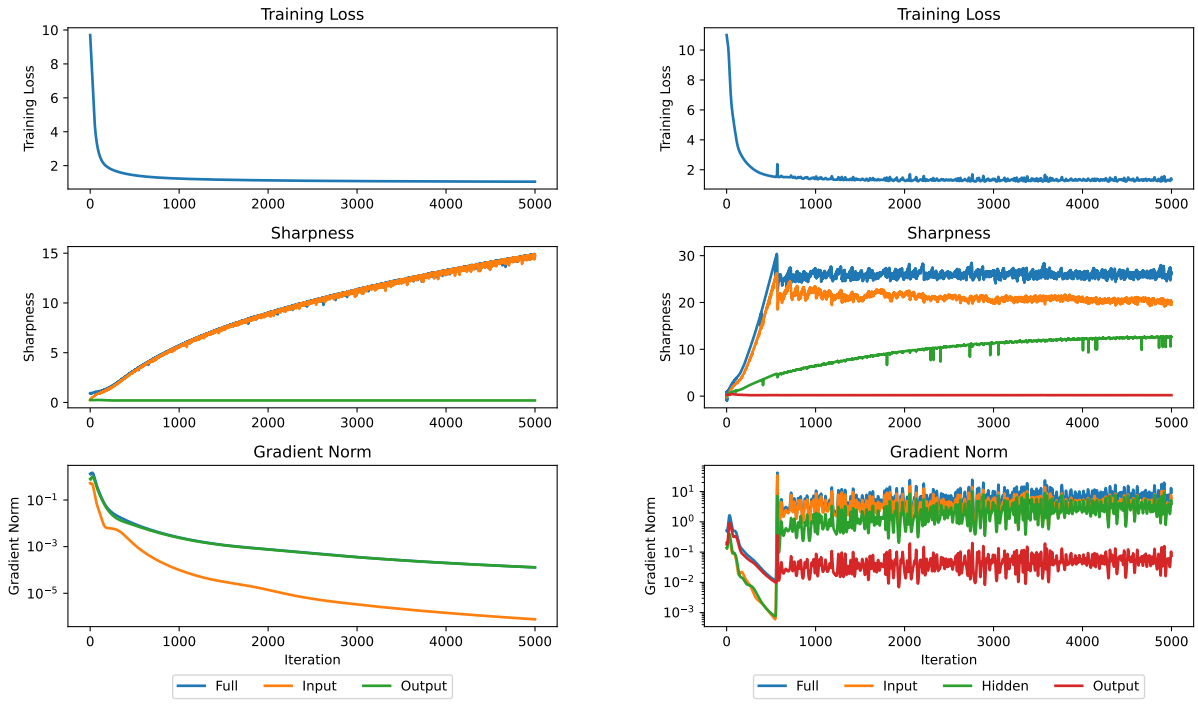
(a) 50 Hidden Units

(b) 100 Hidden Units

Figure 3.4: Edge of Stability compared across different network widths with tanh activation functions using a common stepsize of $\frac{2}{25}$. Each network has 3 layers, and all layers have the same number of hidden units within each trained network. The data used is synthetic regression data.

that $\frac{1}{\lambda_t}$ leads to very slow convergence compared to $\frac{1}{\lambda_0}$ (where λ_0 is the initial sharpness). Note that in Figure 3.7, λ_0 refers to using a step size of $\frac{1}{\lambda_0}$ and λ_t refers to using a step size of $\frac{1}{\lambda_t}$. We also include two constant step sizes in our experiment.

Motivated by the results of the previous section, one might consider using different step sizes for different layers. An obvious approach that takes the $\frac{1}{\lambda_t}$ step size scheme one step further is to consider using the sharpness of each layer in setting the step size. In particular, if the sharpness of layer i on iteration t is $\lambda_{i,t}$, we update the parameters of weight matrix W_i and bias vector b_i with step size $\frac{1}{\lambda_{i,t}}$. This update would be performed for all layers on each step of gradient descent. In our plots, "layer λ" refers to setting the step size of each layer individually with this update scheme. In Figure 3.8, we compare this step size scheme to a backtracking line search (labeled as "LS" in our plots). For all experiments using a backtracking line search in this thesis, we make use of the stochastic line search implementation in Vaswani et al. [2019]. This line search uses one step size and is not done layer-wise. Finally, the λ_0 and λ_t step sizes are the same schemes as used earlier. Note that we are plotting the number of iterations and not the convergence time, and that different step size schemes don't necessarily have iterations with the same time complexity. However, our goal here is to not necessarily give a fast method of convergence, but to illustrate possible ways to select the step size when training neural networks. Finding efficient ways to do so is beyond the scope of the



(a) 2 layers

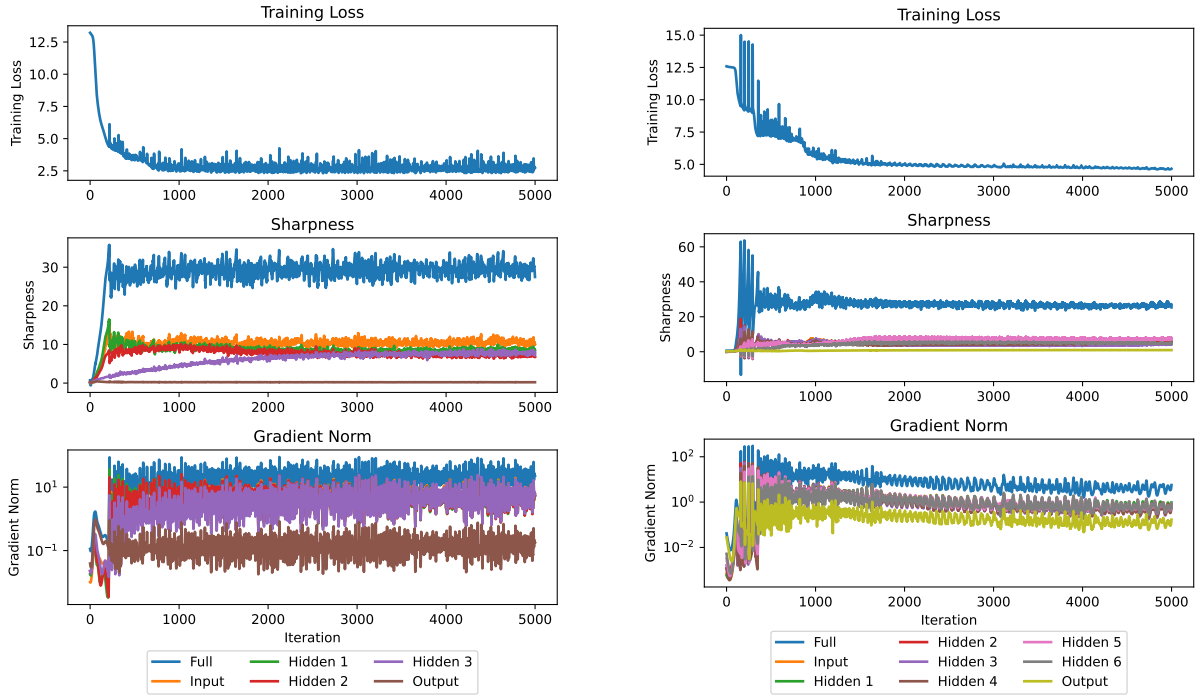
(b) 3 layers

Figure 3.5: Edge of Stability compared across a varying number of layers with tanh activation functions using a common stepsize of $\frac{2}{25}$. All networks trained have 10 hidden units in all layers. The data used is synthetic regression data.

experiments presented in this section.

As we can see in Figure 3.8, for the synthetic data experiment, the layer-wise step size of $\frac{1}{\lambda_{i,t}}$ is clearly too large. However, for the CIFAR-10 experiment, the layer-wise step size seems to make more progress per iteration in decreasing the loss than the other three step size techniques. An obvious and very simple adjustment is to scale the step size $\frac{1}{\lambda_{i,t}}$ of each layer by a constant factor, i.e., to use $\frac{1}{c\lambda_{i,t}}$ for some $c > 0$. In Figure 3.9, we do exactly this and set $c = 2$. We keep all other parameters the same as in Figure 3.8. This leads to significant improvement in the convergence on the synthetic experiments, but leads to much worse performance on the CIFAR-10 dataset. See the Appendix for similar experiments with sigmoid activation functions instead.

One issue with computing the sharpness per layer is that it may be computationally prohibitive for larger datasets or models, so some form of approximation may be required instead. However, another technique for choosing the step size by layer may be both more efficient and lead to faster convergence. Some attempts have been made to explore layer-wise step sizes. A recent paper discusses using a method called learning rate grafting to set the step size of neural networks by layer [Agarwal et al., 2022]. This involves combining the direction of a step from one optimizer and the magnitude of the step from another optimizer to update iterates. The use of layer-wise step sizes warrants further exploration, and could potentially lead to gains



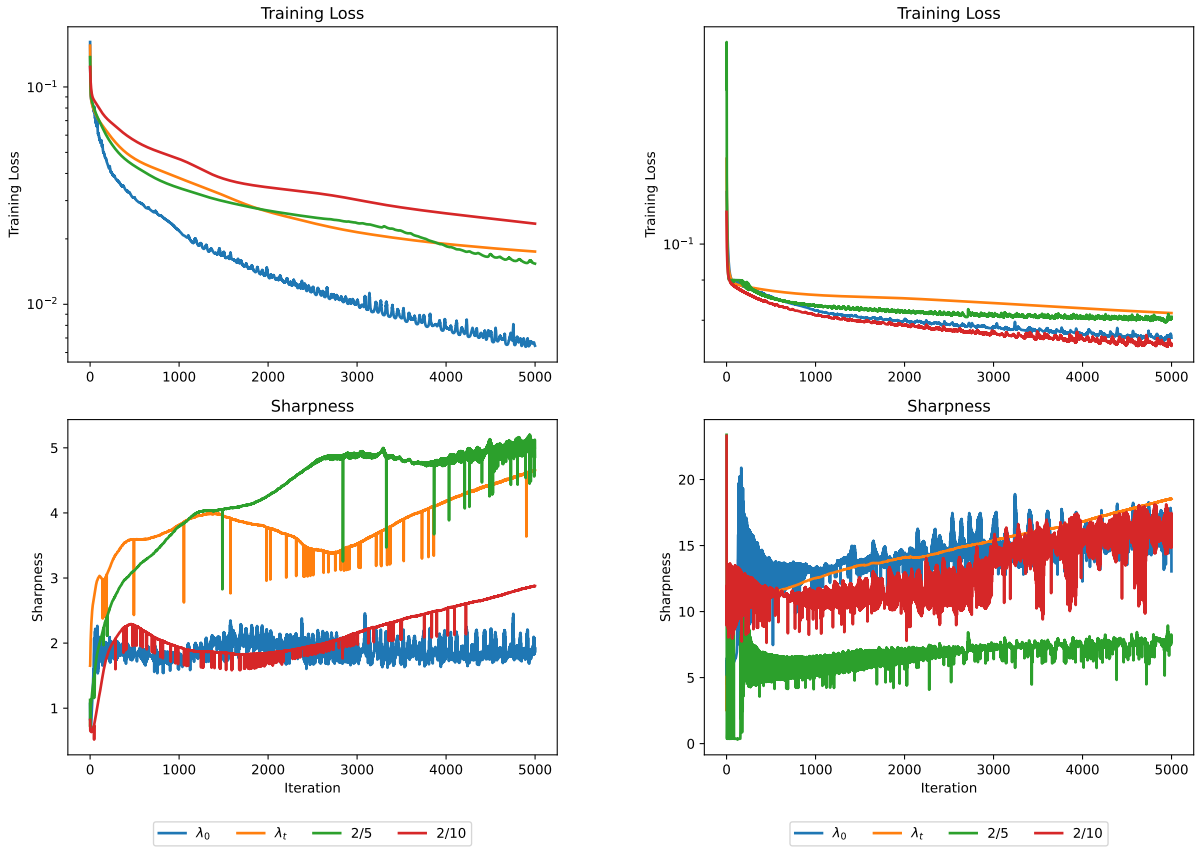
(a) 5 layers

(b) 8 layers

Figure 3.6: Edge of Stability compared across a varying number of layers with tanh activation functions using a common stepsize of $\frac{2}{25}$. All networks trained have 10 hidden units in all layers. The data used is synthetic regression data.

over the more commonly used global step sizes.

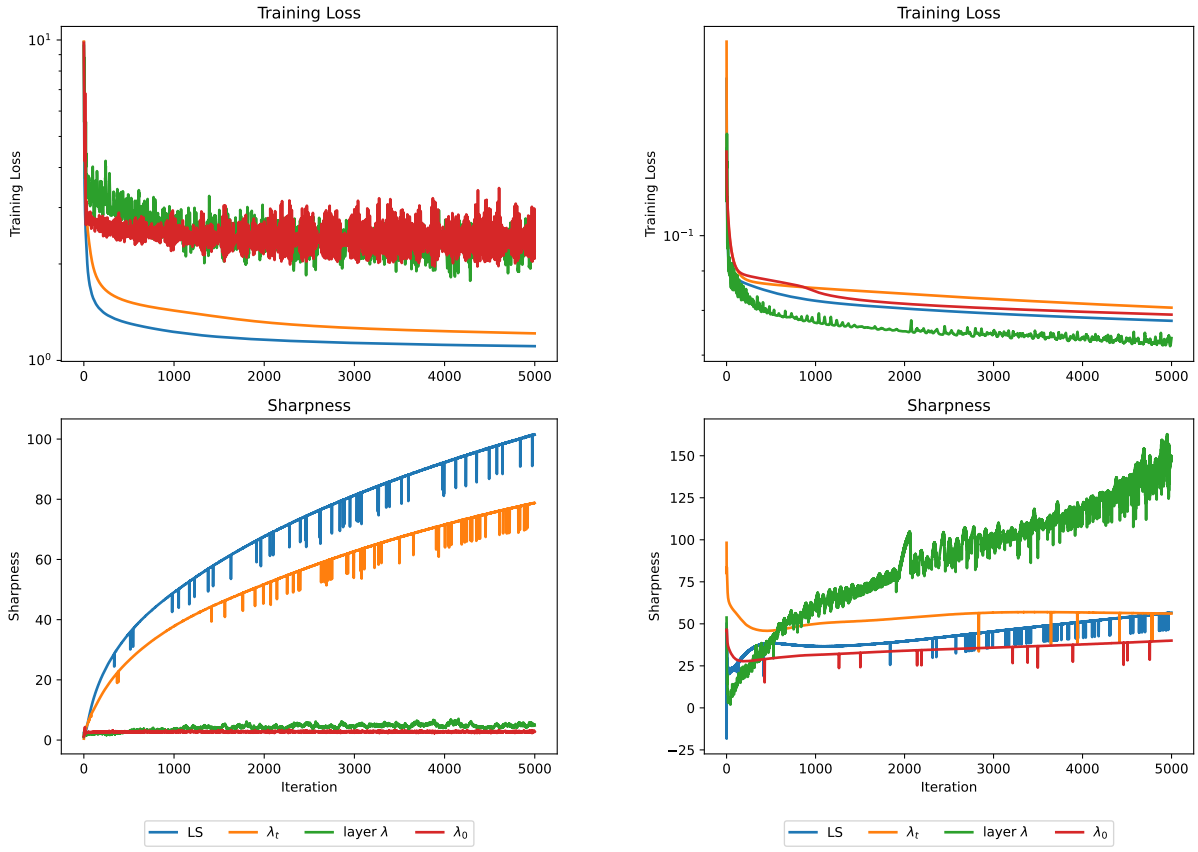
In this chapter we showed experimentally how various architecture choices can lead to possibly different sharpness trajectories for each individual layer. Motivated by these experiments, we briefly discussed the use of layer-wise step sizes. We compared the convergence of step size schedules using the same step size for all layers to that of a layer-wise step size scheme (which makes use of the layer-wise sharpness values).



(a) MNIST

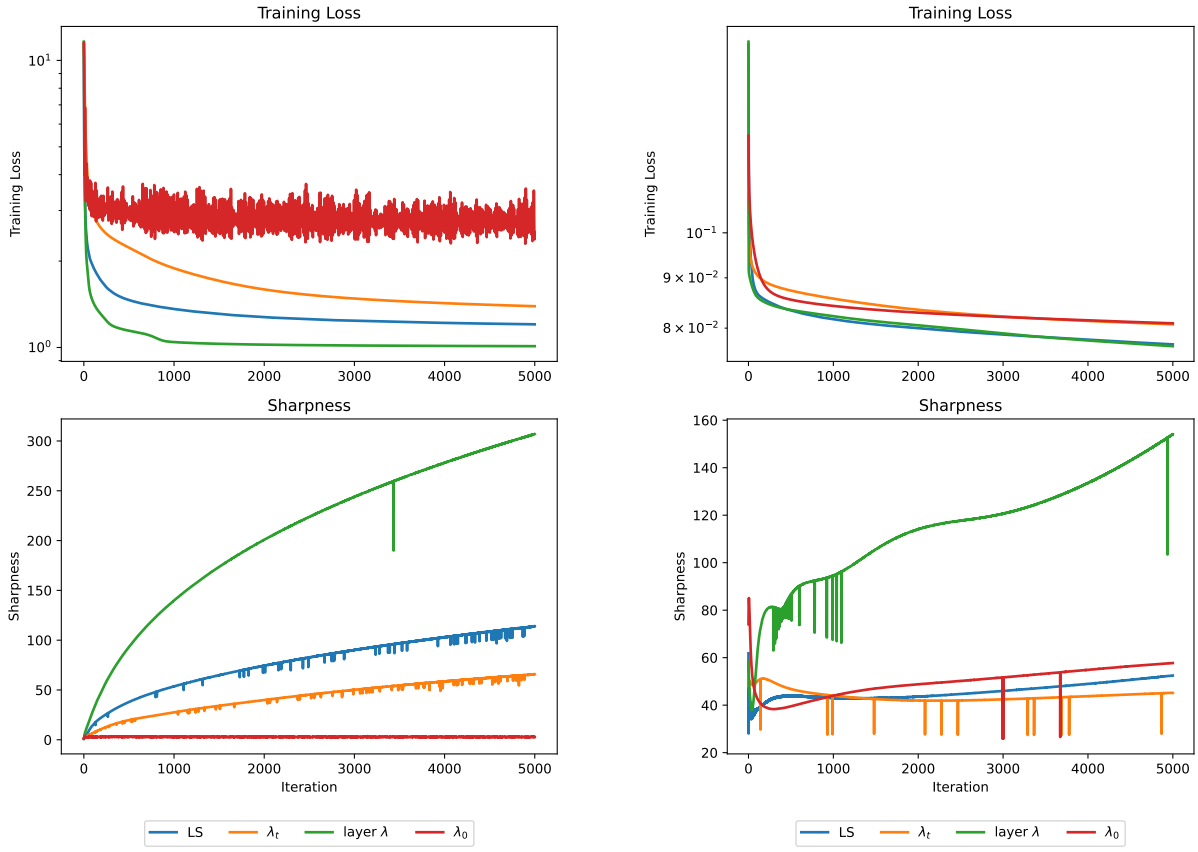
(b) CIFAR-10

Figure 3.7: Convergence for different step sizes and datasets, with ReLU activations.



(a) Synthetic Data **(b)** CIFAR-10

Figure 3.8: Convergence for different step sizes and datasets, with tanh activations.



(a) Synthetic Data **(b)** CIFAR-10

Figure 3.9: Convergence for different step sizes (scaled) and datasets, with tanh activations.

Chapter 4

Stepsize Selection

The focus of the previous chapter was on the EOS from a layerwise perspective, and how one might use layer-wise step sizes to improve training speed of neural networks. In particular, how different step size choices impact training and sharpness. However, a related question is understanding how large one can set the step size before gradient descent diverges entirely (and the loss goes off to infinity). This leads to the following question:

Question: How do various parameters of a neural network architecture (for MLPs) affect the largest possible step size that we can use during training without gradient descent diverging?

4.1 Maximal Step Size Algorithm

In this section, we attempt to shed some light on this question. To help us answer this question, we make use of the algorithm given in Figure 4.1. This algorithm computes the approximate maximal step size that can be selected for gradient descent before it diverges within T epochs, for some given model architecture. Starting from a step size of η_0, the algorithm gradually increases the step size by a factor β until gradient descent diverges. Note that each time the step size is increased, we reset the model so that the weights are reinitialized. This is to ensure we aren't warm starting the model each time. For each model architecture, R runs are performed, and the average maximal step size is then calculated across the R runs.

We note that the work by Gilmer et al. [2021] also looks at how the architecture and choice of step size impact whether gradient descent will diverge or not. However, they look at larger architectures and focus more on how step size selection (without gradient descent diverging) depends on the sharpness at initialization. They observe that the way the weights are initialized does play a role in what step sizes lead to divergence. However, for this thesis we will instead focus on how varying parameters of the network architecture impact which step sizes lead to divergence after a small number of training iterations.

Algorithm 2 Maximal Step Size Selection

Input: initial stepsize η_0, increase factor $\beta > 1$, number of epochs T, runs R

```
for i = 1, ..., R do
    η = η_0
    DIVERGED = False
    while not DIVERGED do
        Initialize new model
        for j = 1, ..., T do
            Perform step of gradient descent
            if gradient descent loss diverges on iteration j (defined in caption) then
                DIVERGED = True
                break
            end if
        end for
        if DIVERGED == False then
            η = β * η
        end if
    end while
    η_i = η
end for
```

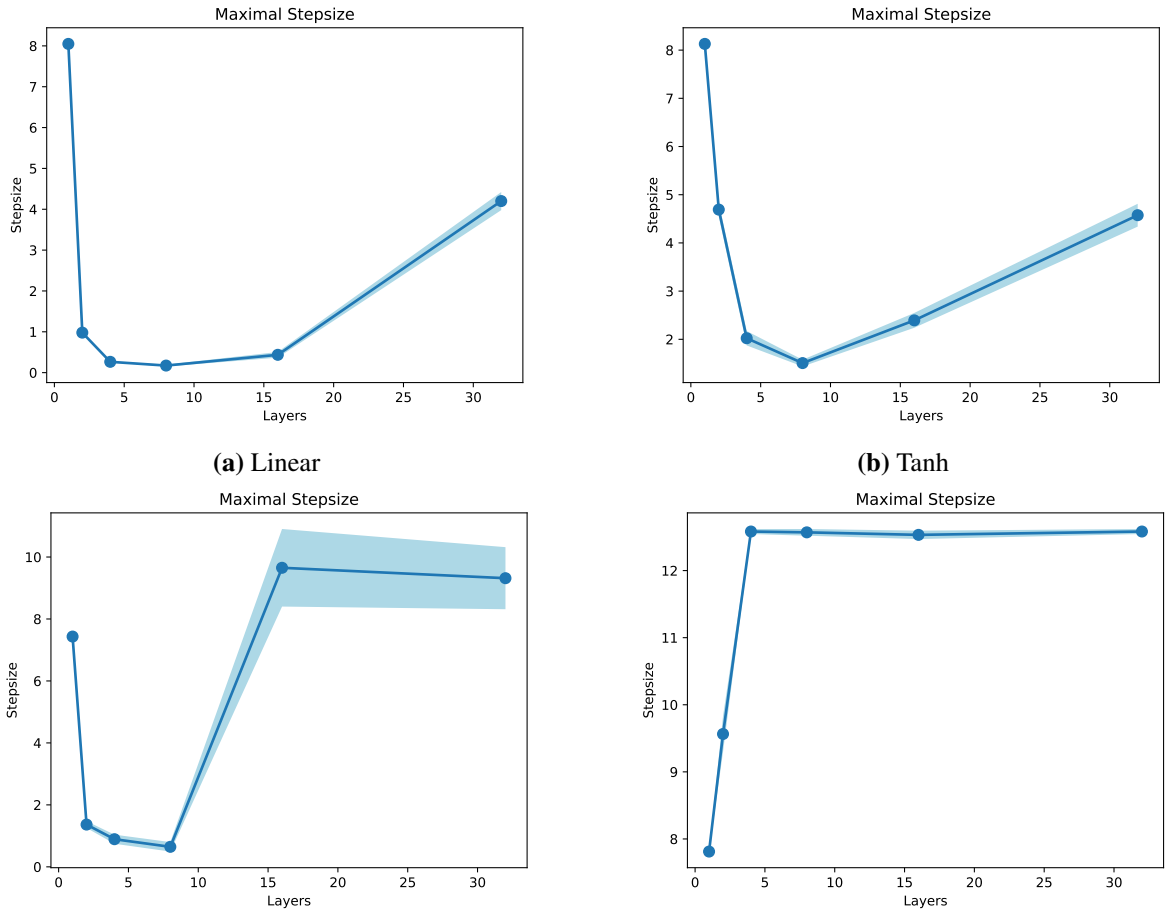
Return:

$\eta_{avg} = \text{mean}\{\eta_i : i = 1, \ldots, R\}$

Figure 4.1: Note that in the `if` statement defined within the while loop, we define gradient descent as diverging on iteration j if the loss computed on iteration j is either not a number (NAN) or floating point infinity (as computed in the code).

4.2 Maximal Step Size Results

In this section we make use of the algorithm in Figure 4.1 to test how various architecture choices affect the average (across R runs) maximal step size that can be chosen before full batch gradient descent diverges within some fixed number of iterations. In Figure 4.2 we compare the maximal step sizes computed across models with a varying number of layers, using a different activation function for each subfigure. In Figure 4.3 we instead compare across a varying number of hidden units, again doing so for different activation functions. We similarly perform experiments varying the number of training examples in Figure 4.4. Finally, in Table 4.1, we compare the maximal step size across different activation functions. In the first row "Linear" refers to a network with no non-linear activation functions used in any layers. Note that all of these experiments use synthetic regression data. The number of runs R is set to 10, and the number of iterations T is set to 100. The increase factor β is set to 1.01 and the starting step size η_{init} is set to 0.1. This starting step size is hand tuned, since we don't know the exact point of divergence, and it's unclear what a good starting threshold is across a variety of architectures because in some sense this is the question we are attempting to



(a) Linear

(b) Tanh

(c) ReLU

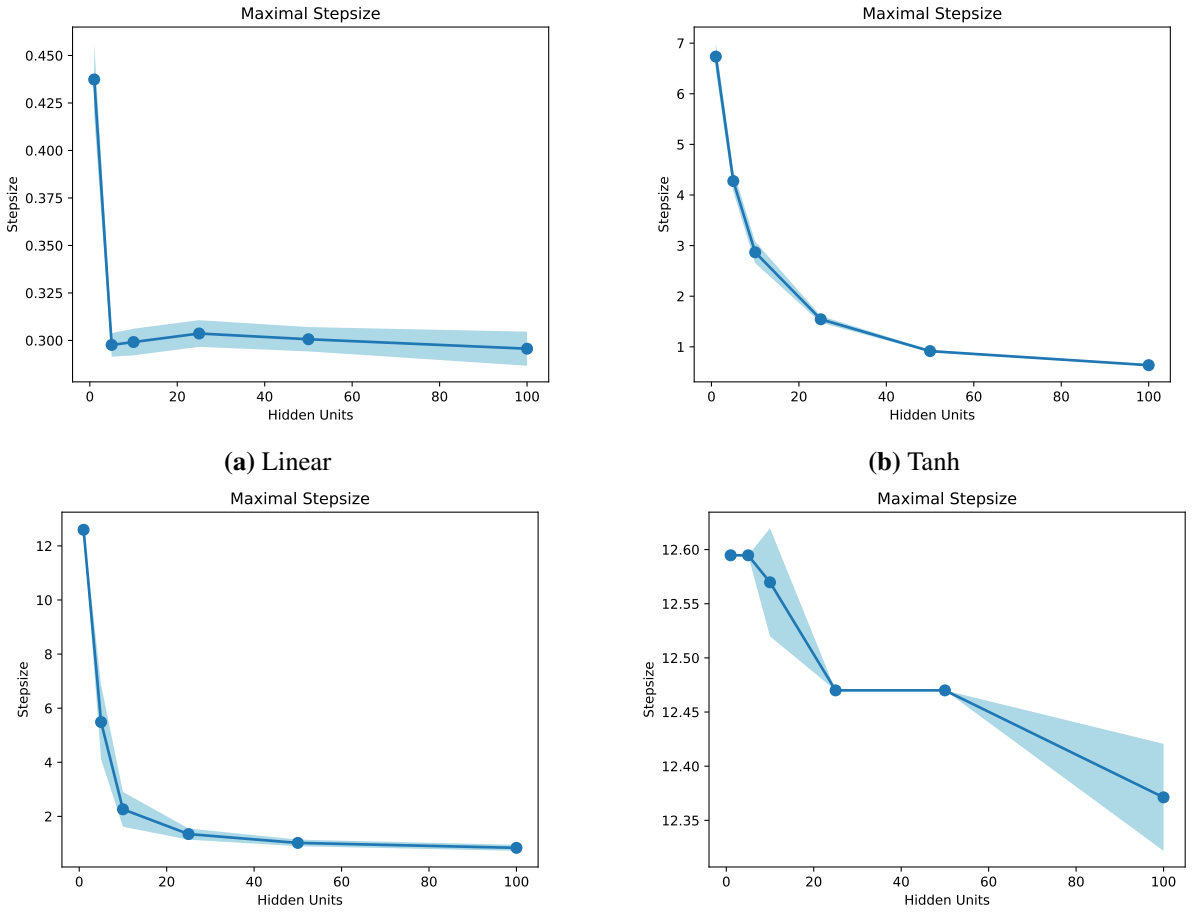
(d) Sigmoid

Figure 4.2: Comparing the maximum step size that can be selected before full batch gradient descent diverges across a differing number of network depths for different activation functions, with each network having 10 hidden units per layer and 100 training examples used for training. Note that the data used is synthetic regression data.

answer. We make the following observations in our experiments:

- For networks with linear, ReLU, and tanh activations, we see from Figure 4.2 that for networks with fewer layers, larger step sizes are possible. As more layers are added, smaller step sizes must be used to ensure gradient descent doesn't diverge early in training. However, at some point (in our experiment, a 16 layer network), larger step sizes are again possible. When using sigmoid activations instead, we observe a different trend as we vary the number of network layers. Instead, we see a more monotonic trend, where more layers corresponds to a larger maximal step size, before leveling out after 4 layers. We note that it is not obvious if smaller or larger step sizes should generally be required for deeper networks, and our results seem to corroborate this.

- Based on Figure 4.3, for wider networks we are forced to use smaller step sizes. This is in line with



(a) Linear

(b) Tanh

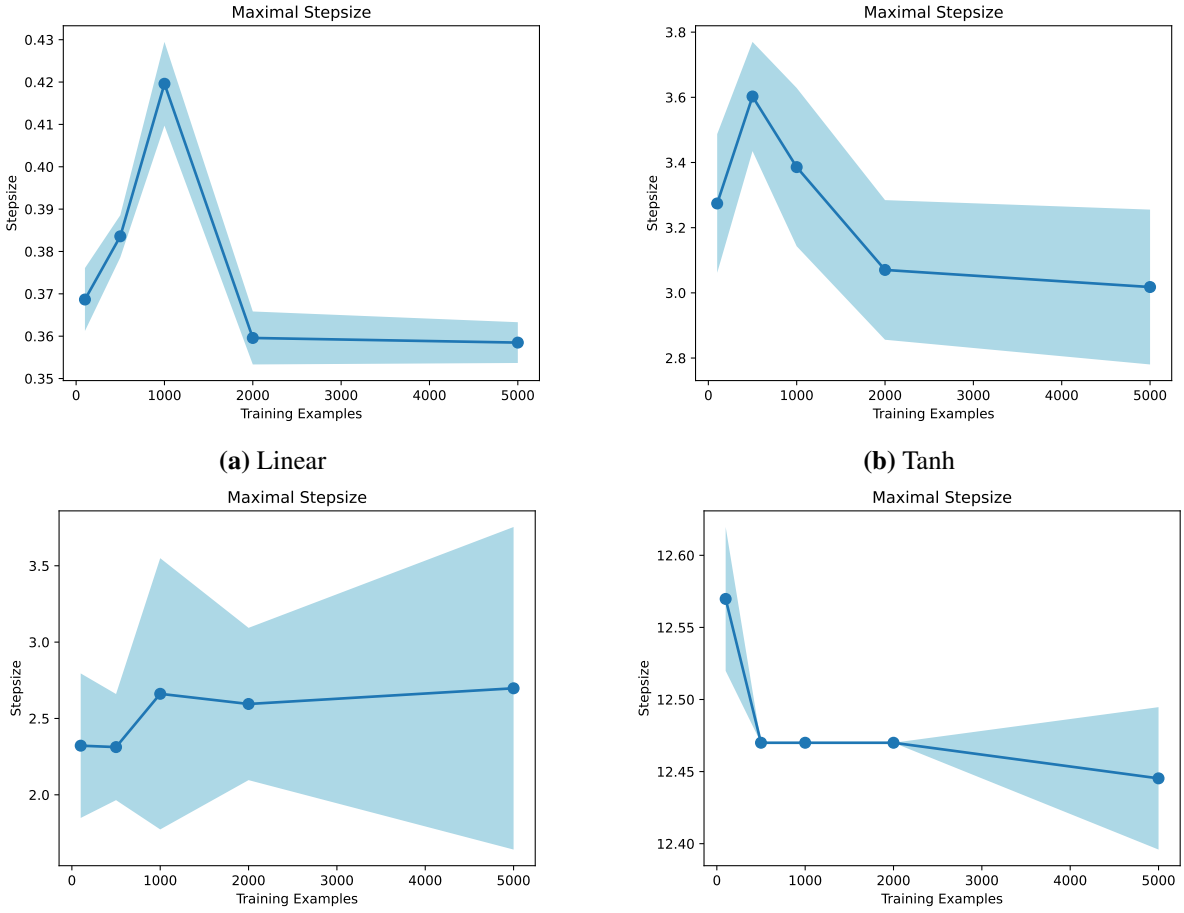
(c) ReLU

(d) Sigmoid

Figure 4.3: Comparing the maximum step size that can be selected before full batch gradient descent diverges across a differing number of hidden units per layer for different activation functions, with each network having 3 layer and 100 training examples used for training. Note that the data used is synthetic regression data.

Table 4.1: Comparing the maximum step size that can be selected before full batch gradient descent diverges across different activation functions, with each network having 3 layers and 10 hidden units per layer trained on 100 training examples.

Activation	Mean Maximal Stepsize
Linear	0.38433
Tanh	3.12765
RELU	2.26166
Sigmoid	12.58223



(a) Linear

(b) Tanh

(c) ReLU

(d) Sigmoid

Figure 4.4: Comparing the maximum step size that can be selected before full batch gradient descent diverges across a varying number of training examples for different activation functions, with each network having 3 layers and 10 hidden units per layer. Note that the data used is synthetic regression data.

what we would expect, that wider networks require smaller step sizes to ensure convergence. Unlike the case when we varied the number of layers, we observe that the maximal step size trend is mostly monotonic across all types of activation functions tested, where more hidden units corresponds to a smaller maximal step size. One notable difference between the subfigures is that for linear and sigmoid activations, the difference in the allowable maximal stepsize for a varied number of hidden units is very minimal (less than a difference 0.3 between 1 hidden unit and 100 hidden units in both cases). In other words, varying the number of hidden units for these cases made almost no difference.

- We note that varying the training set size seemed to have the least impact on the step size range that lead to divergence, based on the results given in Figure 4.4. Notably, this trend held across all our choices of activation functions. It would be interesting to further explore why the training set size had

Table 4.2: Comparing the maximum step size that can be selected before gradient descent with batch size 1 diverges across different activation functions, with each network having 3 layers and 10 hidden units per layer trained on 100 training examples.

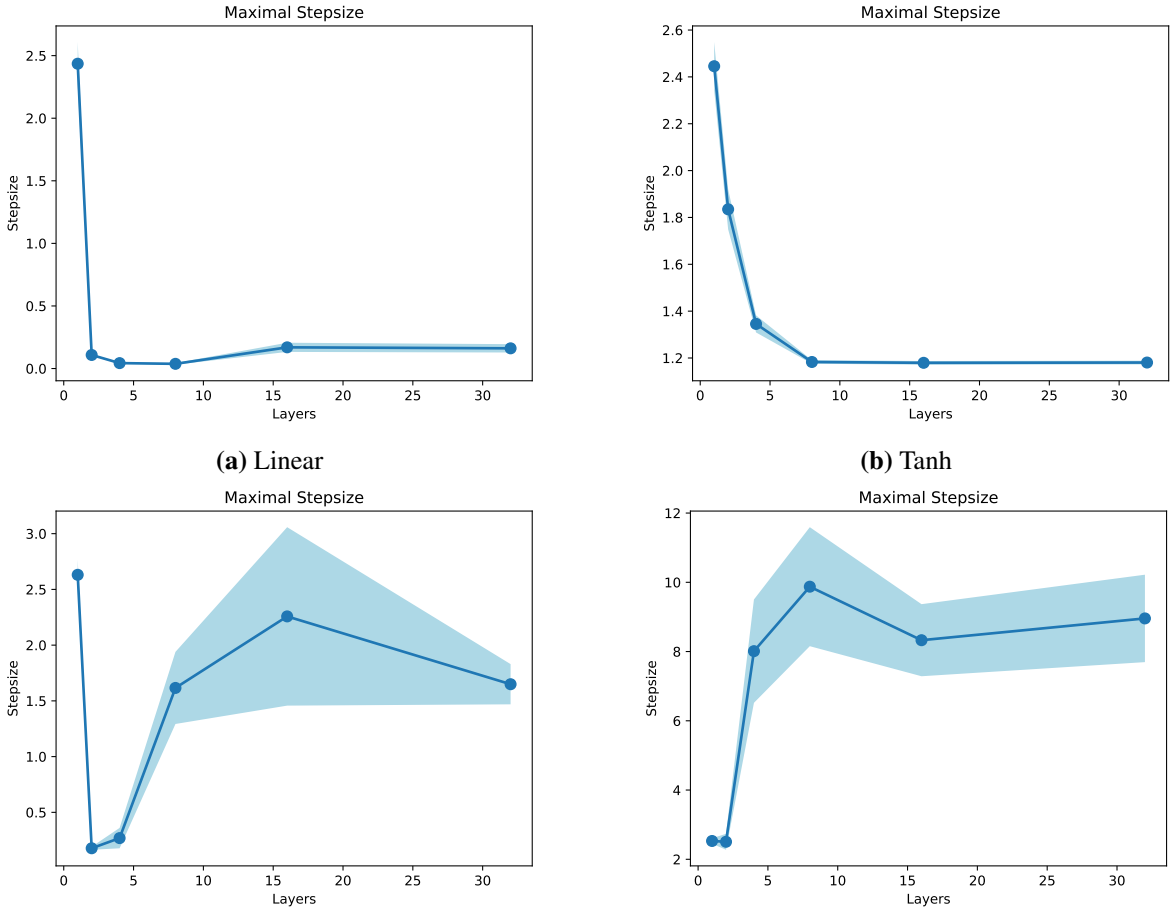
Activation	Maximal Stepsize
Linear	0.05842
Tanh	1.44604
RELU	0.19952
Sigmoid	9.17098

a smaller impact on the maximal step size, as it isn't obvious why this is the case.

- Finally, in the case that we vary the activation functions in Table 4.1, we see that the step sizes leading to divergence can be vastly different between differing choices of activation functions. This is not entirely surprising as the activation functions used dramatically affected the experimental results given in Chapter 3 as well.

In Figures 4.5 through 4.7, as well as Table 4.2, we perform similar experiments as above except with a batch size of one (so we are doing SGD rather than full batch gradient descent). For these experiments, we use an initial step size of 0.01 instead of 0.1. We observe similar trends to that of the full batch case. However, as expected the maximal step sizes are smaller since SGD generally requires smaller step sizes than full batch gradient descent for the same problem.

Overall, the goal of these experiments is to help give some insights into the potential impacts that various architecture choices can have on appropriate step size selection for gradient descent. We give further experimental results exploring the maximal step size that leads to divergence for gradient descent in the Appendix.



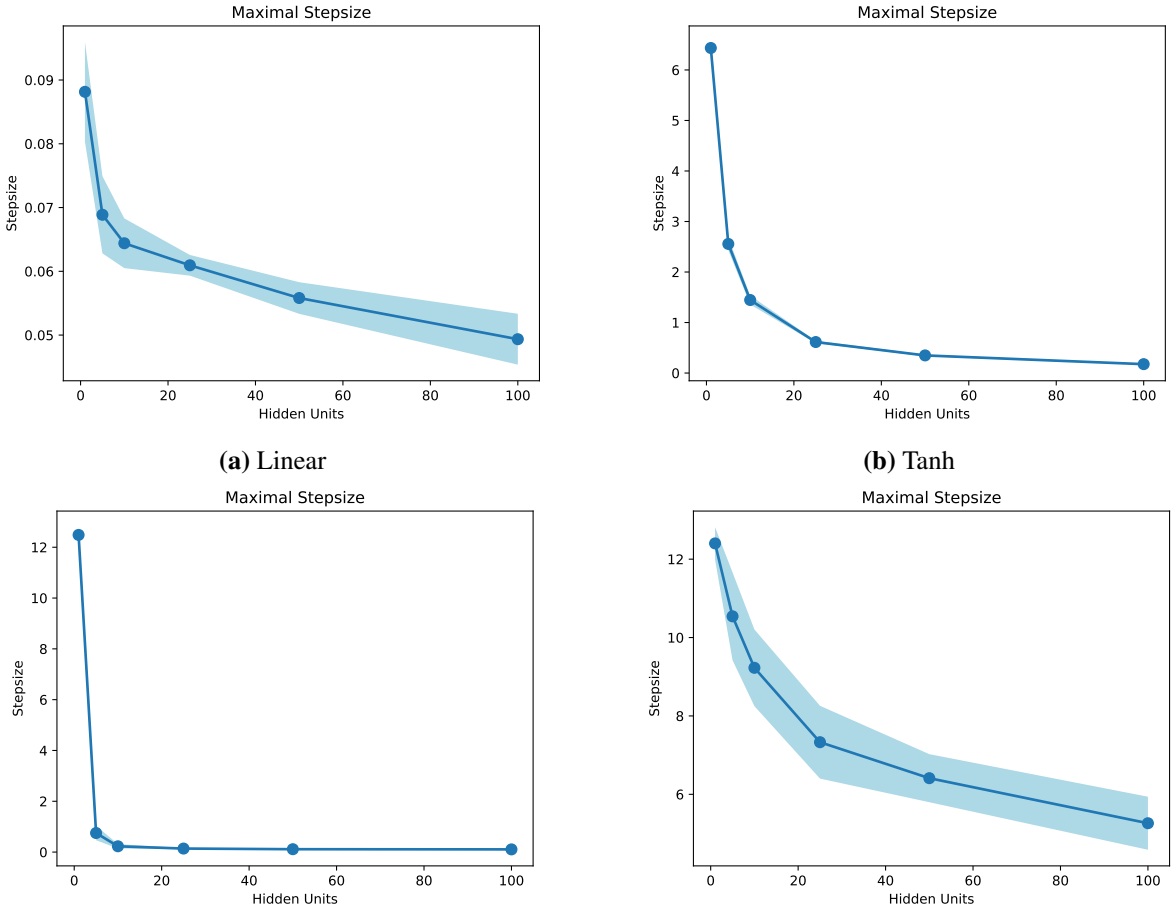
(a) Linear

(b) Tanh

(c) ReLU

(d) Sigmoid

Figure 4.5: Comparing the maximum step size that can be selected before gradient descent with batch size 1 diverges across a differing number of network depths for different activation functions, with each network having 10 hidden units per layer and 100 training examples used for training. Note that the data used is synthetic regression data.



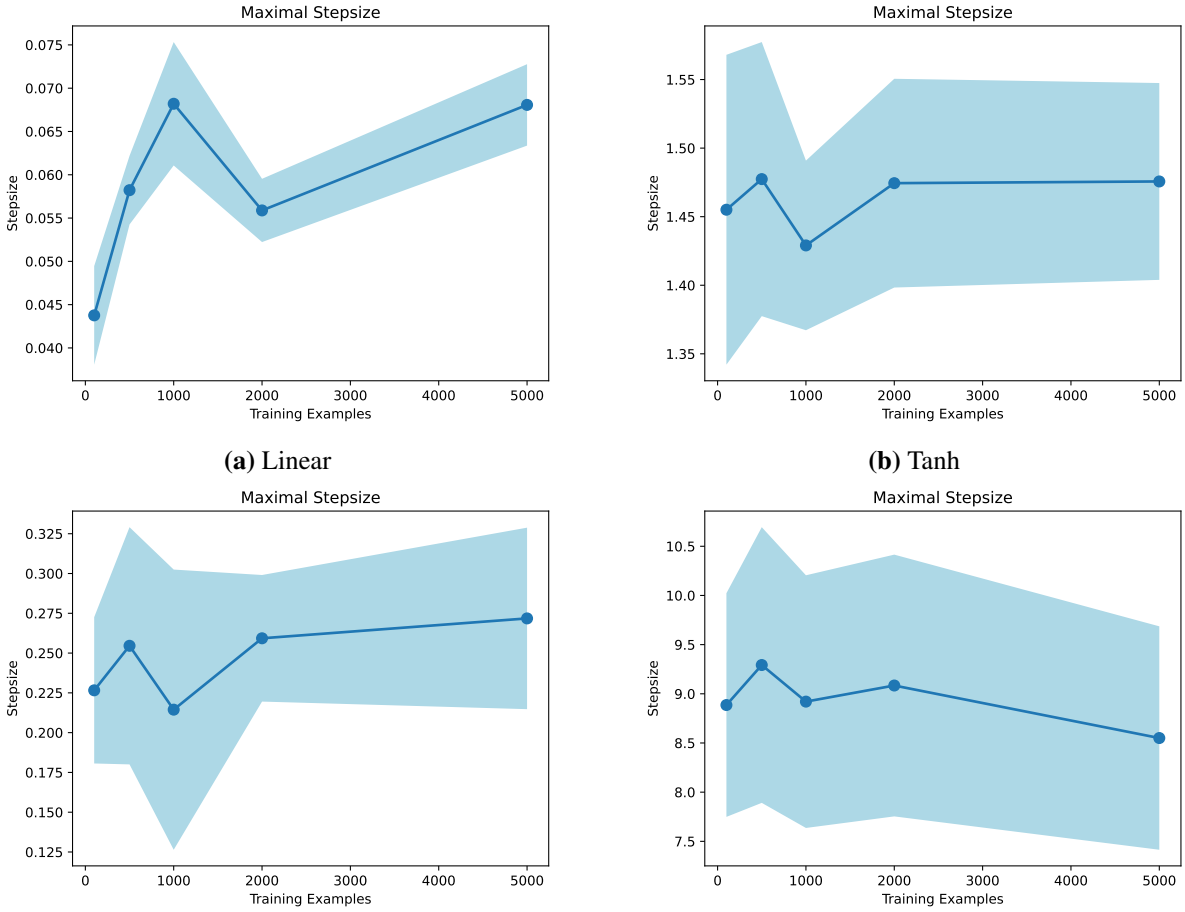
(a) Linear

(b) Tanh

(c) ReLU

(d) Sigmoid

Figure 4.6: Comparing the maximum step size that can be selected before gradient descent with batch size 1 diverges across a differing number of hidden units per layer for different activation functions, with each network having 3 layer and 100 training examples used for training. Note that the data used is synthetic regression data.



(a) Linear

(b) Tanh

(c) ReLU

(d) Sigmoid

Figure 4.7: Comparing the maximum step size that can be selected before gradient descent with batch size 1 diverges across a varying number of training examples for different activation functions, with each network having 3 layers and 10 hidden units per layer. Note that the data used is synthetic regression data.

Chapter 5

Conclusion

In this thesis we examined how the sharpness of individual layers of deep neural networks behave at the EOS. We experimentally show that progressive sharpening and EOS-like behavior occur for individual layers, and observe that different layers can have contrasting sharpness behavior. Interestingly, we also observe EOS-like behavior of the gradient norms of each layer. We briefly explore how layer-wise step sizes could potentially lead to gains in training convergence rates, and comment on existing techniques for setting layer-wise step sizes. Finally, we highlight how certain neural network architecture choices, such as the number of network layers and type of activation function used, can greatly affect the maximal constant step size that can be used before gradient descent diverges. Based on our results, we discuss how certain architecture choices may have more impact on this maximal step size than others.

A natural extension of this work would be looking at other types of neural networks that were examined in Cohen et al. [2021], such as convolutional neural networks or transformers, and applying a similar analysis to them as we did with MLPs. Transformers may be of particular relevance as they are a newer architecture that is being widely used, but that still very much lack theoretical grounding. Gaining a better understanding of them could be of practical benefit. Also extending this work to larger models or datasets could lead to interesting insights. This thesis mainly focuses on synthetic data regression data, and the trends we observe with this data may differ from those seen on real world data. Another line of work would be how neural network weight initialization plays a role in training as a whole, particularly its impact on network sharpness. The work by Iyer et al. [2023] looks briefly at how weight initialization affects network sharpness, but their work does not examine the sharpness layer-wise unlike this thesis. Further exploring layer-wise step size schemes could also be potentially fruitful, as this line of research has been sparsely examined to date. Also, note that our experiments in Chapter 3 are done in the full batch setting, which is not generally used in practice. Thus, extending the experiments in this chapter to the stochastic setting may be of interest. It would be interesting to see what similarities, if any, the full batch setting has with the stochastic setting in terms of our layer-wise sharpness analysis. As well, note that we only use the MSE loss function for our experiments, and they could easily be extended to other commonly used loss functions. Also seeing how eigenvalues other than the top eigenvalue of each layer change over the course of training may be insightful

as well. Finally, limited research has explored the EOS for adaptive optimization methods, such as the Adam optimizer [Cohen et al., 2022]. In particular, Adam is often the optimization algorithm of choice in practice when training neural networks, so applying this work to Adam could be useful in practice. We hope that this thesis takes one more step towards helping researchers gain a better understanding neural network training with gradient descent.

Bibliography

Agarwal, N., Anil, R., Hazan, E., Koren, T., and Zhang, C. (2022). Learning rate grafting: Transferability of optimizer tuning. → page 19

Agarwala, A., Pedregosa, F., and Pennington, J. (2022). Second-order regression models exhibit progressive sharpening to the edge of stability. *arXiv preprint arXiv:2210.04860*. → page 8

Ahn, K., Bubeck, S., Chewi, S., Lee, Y. T., Suarez, F., and Zhang, Y. (2022a). Learning threshold neurons via the" edge of stability". *arXiv preprint arXiv:2212.07469*. → page 10

Ahn, K., Jadbabaie, A., and Sra, S. (2023). How to escape sharp minima. *arXiv preprint arXiv:2305.15659*. → page 10

Ahn, K., Zhang, J., and Sra, S. (2022b). Understanding the unstable convergence of gradient descent. In *ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, pages 247–257. PMLR. → page 8

Andriushchenko, M. and Flammarion, N. (2022). Towards understanding sharpness-aware minimization. In *ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, pages 639–668. PMLR. → page 10

Arora, S., Li, Z., and Panigrahi, A. (2022). ICML 2022. volume 162 of *Proceedings of Machine Learning Research*, pages 948–1024. PMLR. → page 8

Barzilai, J. and Borwein, J. M. (1988). Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis*, 8(1):141–148. → page 2

Cauchy, L. A. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. → page 2

Chen, L. and Bruna, J. (2022). On gradient descent convergence beyond the edge of stability. *arXiv preprint arXiv:2206.04172*. → page 9

Cohen, J. M., Ghorbani, B., Krishnan, S., Agarwal, N., Medapati, S., Badura, M., Suo, D., Cardoze, D., Nado, Z., Dahl, G. E., et al. (2022). Adaptive gradient methods at the edge of stability. *arXiv preprint arXiv:2207.14484*. → page 34

Cohen, J. M., Kaur, S., Li, Y., Kolter, J. Z., and Talwalkar, A. (2021). Gradient descent on neural networks typically occurs at the edge of stability. In *ICLR 2021*. → pages 4, 5, 6, 8, 12, 14, 17, 33

Damian, A., Nichani, E., and Lee, J. D. (2023). Self-stabilization: The implicit bias of gradient descent at the edge of stability. In *ICLR 2023*. → page 10

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT 2019*, pages 4171–4186. → page 3

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR 2021*. → page 3

Du, J., Yan, H., Feng, J., Zhou, J. T., Zhen, L., Goh, R. S. M., and Tan, V. Y. F. (2022). Efficient sharpness-aware minimization for improved training of neural networks. In *ICLR 2022*. → page 10

Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JLMR*, 12:2121–2159. → page 3

Even, M., Pesme, S., Gunasekar, S., and Flammarion, N. (2023). (s) gd over diagonal linear networks: Implicit regularisation, large stepsizes and edge of stability. *arXiv preprint arXiv:2302.08982*. → page 11

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. (2021). Sharpness-aware minimization for efficiently improving generalization. In *ICLR 2021*. → page 10

Friedlander, M. P. and Schmidt, M. (2013). Erratum: Hybrid deterministic-stochastic methods for data fitting. *SIAM J. Sci. Comput.*, 35(4). → page 17

Ghadimi, S. and Lan, G. (2013). Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM J. Optim.*, 23(4):2341–2368. → page 17

Gilmer, J., Ghorbani, B., Garg, A., Kudugunta, S., Neyshabur, B., Cardoze, D., Dahl, G., Nado, Z., and Firat, O. (2021). A loss curvature perspective on training instability in deep learning. *arXiv preprint arXiv:2110.04369*. → page 24

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org. → page 5

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV 2015*, pages 1026–1034. IEEE Computer Society. → page 5

Iyer, G., Hanin, B., and Rolnick, D. (2023). Maximal initial learning rates in deep relu networks. In *International Conference on Machine Learning*, pages 14500–14530. PMLR. → page 33

Kaur, S., Cohen, J., and Lipton, Z. C. (2022). On the maximum hessian eigenvalue and generalization. In *NeurIPS 2022 Workshops*, volume 187 of *Proceedings of Machine Learning Research*, pages 51–65. PMLR. → page 11

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR 2015*. → page 3

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. → page 13

Kumar, S. K. (2017). On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*. → page 5

Kwon, J., Kim, J., Park, H., and Choi, I. K. (2021). ASAM: adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. In Meila, M. and Zhang, T., editors, *ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pages 5905–5914. PMLR. → page 10

Lacoste-Julien, S., Schmidt, M., and Bach, F. (2012). A simpler approach to obtaining an $\mathcal{O}(1/t)$ convergence rate for the projected stochastic subgradient method. *arXiv preprint arXiv:1212.2002*. → page 2

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324. → page 13

Lewkowycz, A., Bahri, Y., Dyer, E., Sohl-Dickstein, J., and Gur-Ari, G. (2020). The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*. → page 10

Li, X., Zhuang, Z., and Orabona, F. (2021). A second look at exponential and cosine step sizes: Simplicity, adaptivity, and performance. In Meila, M. and Zhang, T., editors, *ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pages 6553–6564. PMLR. → page 2

Linnainmaa, S. (1970). *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki. → page 5

Liu, Y., Mai, S., Chen, X., Hsieh, C., and You, Y. (2022). Towards efficient and scalable sharpness-aware minimization. In *CVPR 2022*, pages 12350–12360. IEEE. → page 10

Loshchilov, I. and Hutter, F. (2017). SGDR: stochastic gradient descent with warm restarts. In *ICLR 2017*. → page 2

Ma, C., Kunin, D., Wu, L., and Ying, L. (2022). Beyond the quadratic approximation: the multiscale structure of neural network loss landscapes. *arXiv preprint arXiv:2204.11326*. → page 8

Narkhede, M. V., Bartakke, P. P., and Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artif. Intell. Rev.*, 55(1):291–322. → page 5

Nesterov, Y. (2018). *Lectures on convex optimization*. Springer. → pages 3, 4

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407. → page 3

Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY. → page 5

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. → page 5

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NeurIPS 2017*, pages 5998–6008. → page 3

Vaswani, S., Mishkin, A., Laradji, I. H., Schmidt, M., Gidel, G., and Lacoste-Julien, S. (2019). Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *NeurIPS 2019*, pages 3727–3740. → page 18

Wang, X., Magnússon, S., and Johansson, M. (2021). On the convergence of step decay step-size for stochastic optimization. In *NeurIPS 2021*, pages 14226–14238. → page 2

Wang, Z., Li, Z., and Li, J. (2022). Analyzing sharpness along GD trajectory: Progressive sharpening and edge of stability. In *NeurIPS 2022*. → page 17

Wu, L., Ma, C., and E, W. (2018). How SGD selects the global minima in over-parameterized learning: A dynamical stability perspective. In *NeurIPS 2018*, pages 8289–8298. → page 10

Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. W. (2020). Pyhessian: Neural networks through the lens of the hessian. In *IEEE 2020*, pages 581–590. → pages 12, 13

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*. → page 3

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. In *ICLR 2017*. → page 2

Zhu, X., Wang, Z., Wang, X., Zhou, M., and Ge, R. (2023). Understanding edge-of-stability training dynamics with a minimalist example. In *ICLR 2023*. → page 9

Appendix A

Additional Experimental Results

A.1 Layer-wise Experiments

In this section, we include further experiments exploring the layer-wise sharpness results from Section 3.2, where we instead use a larger step size of $\frac{2}{10}$. See Figures A.1 through A.5. Most of the results are somewhat comparable in this section to those given in Chapter 3. However, there are some differences worth noting:

- For some of the experiments given in Chapter 3, the EOS is not reached in the 5000 iterations we train for using a step size of $\frac{2}{25}$. When using a larger step size of $\frac{2}{10}$, we do see that the EOS is reached within 5000 iterations. This is not surprising as we expect to reach the EOS more quickly with a larger step size.
- In the Chapter 3 we do our layer-wise experiments comparing across different numbers of layers or different numbers of hidden units using tanh activation functions. See Figures 3.3 through 3.6 for these experiments. In this section, we perform similar experiments using ReLU activation functions as well. Unlike the tanh activation experiments, when using ReLU activations we observe that when varying the number of hidden units, the sharpness per layer is comparable within each plot and we don't see the input layer having drastically larger sharpness than the other layers in this case.

A.2 Step Size Experiments

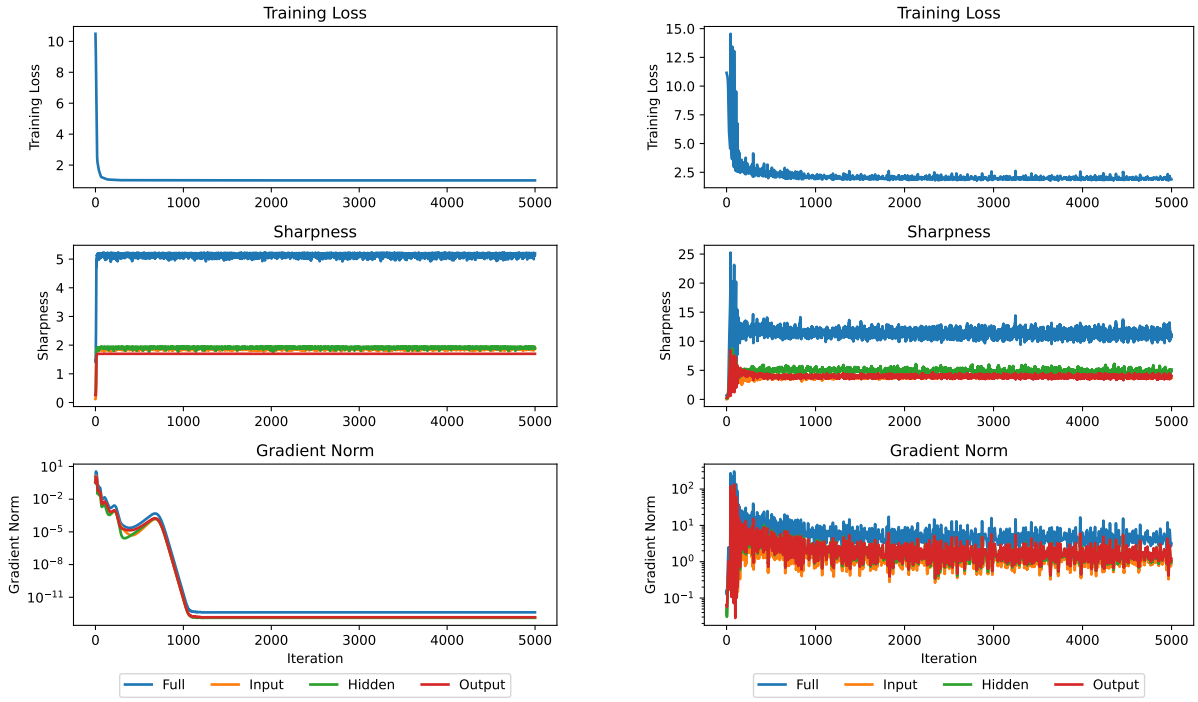
In this section, we provide further experiments using the same layer-wise step size scheme from section 3.3, instead using sigmoid activation functions here. See Figures A.6 and A.7. The results are very similar to that of the experiments using tanh activations in Chapter 3.

A.3 Divergence Experiments

In this section, we include further experiments exploring the maximal step size results from Section 4.2. See Figures A.8 through A.10 for maximal step size experiments using the CIFAR-10 dataset rather than

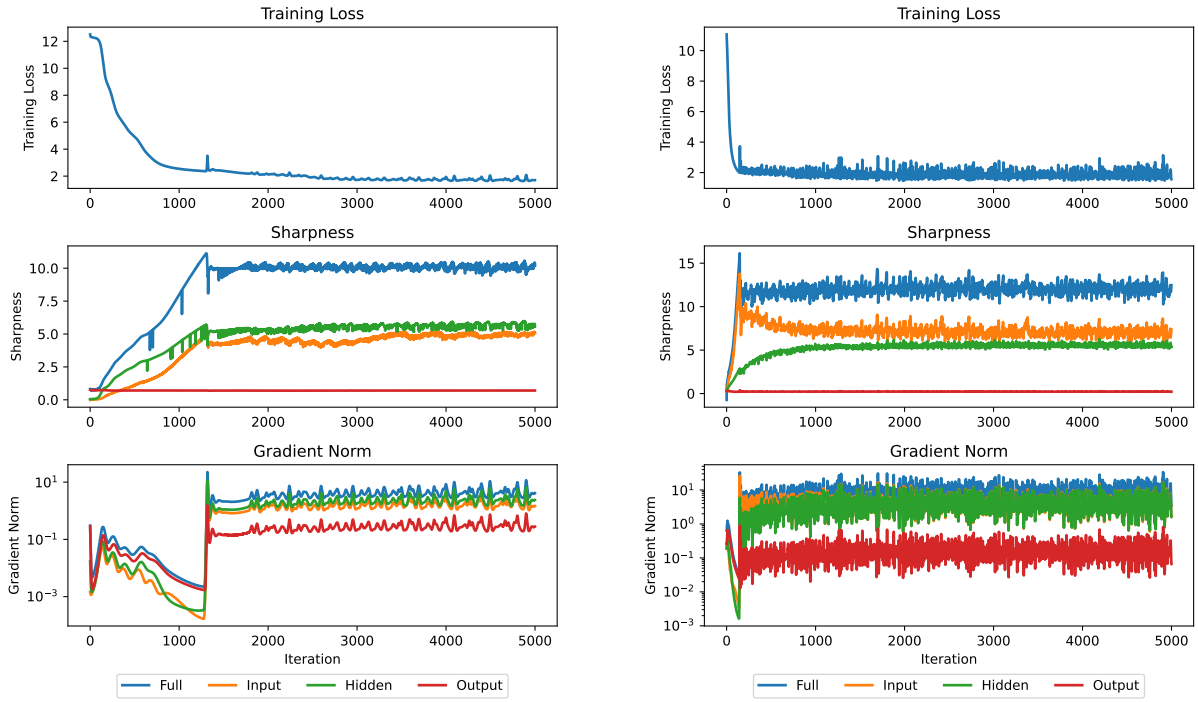
synthetic regression data. Note that unlike the full batch synthetic regression experiments in Section 4.2, the hand picked initial step size for experiments in this section is 0.01 rather than 0.1. Most of the results are similar to that of Chapter 4, with a few notable differences:

- In the linear activation case, when varying the number of hidden units, the allowable step size increases as the number of hidden units increases when using CIFAR-10 data. This is the opposite trend observed in Chapter 4.
- When varying the number of layers with CIFAR-10 data, the allowable step size monotonically increases as the number of layers is increased (except in the ReLU case).



(a) Linear

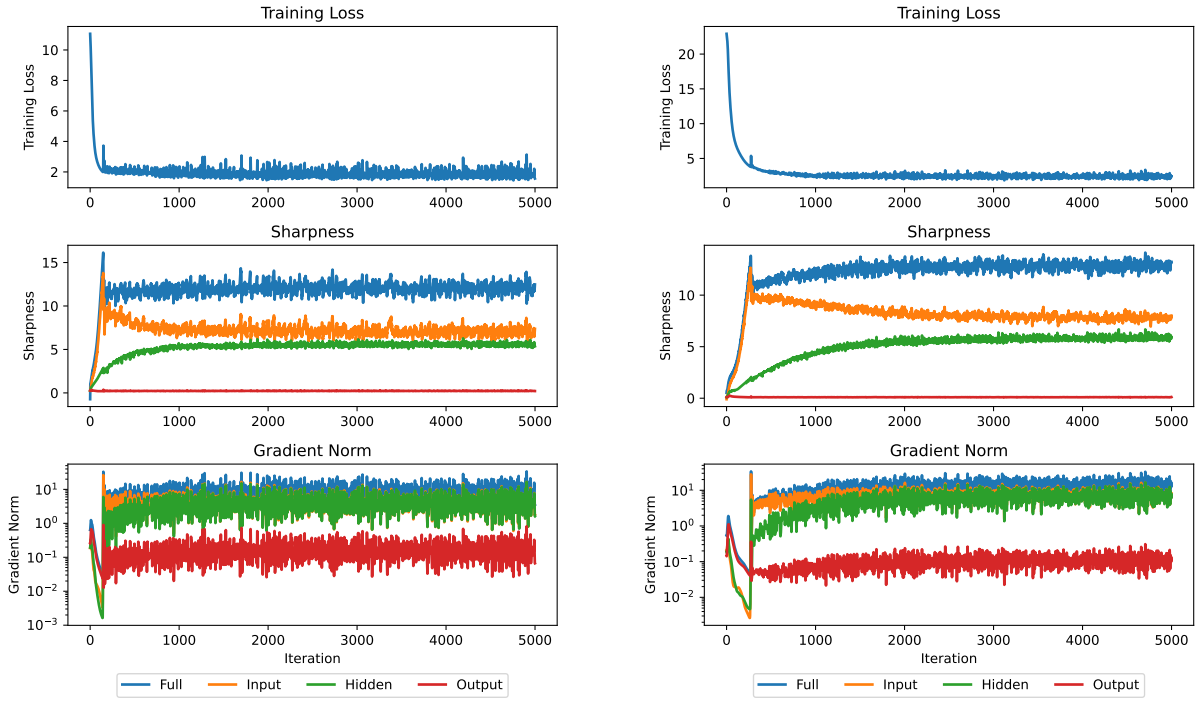
(b) ReLU



(c) Sigmoid

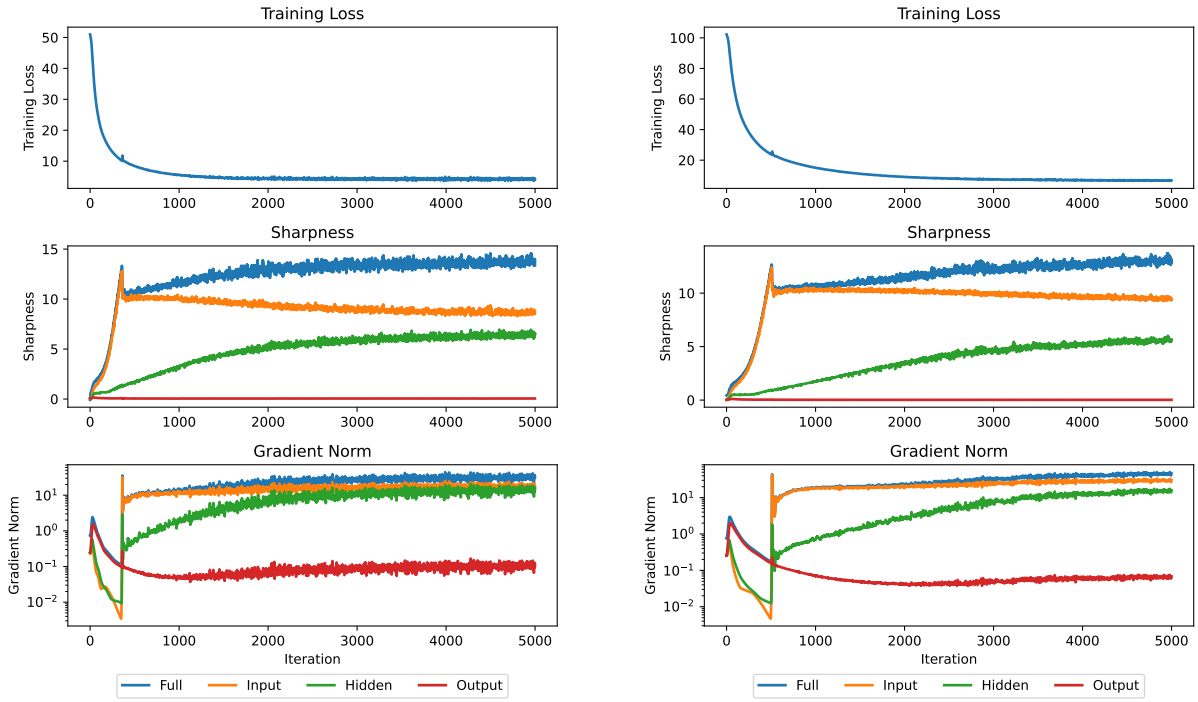
(d) Tanh

Figure A.1: Edge of Stability compared across different activation functions using a common stepsize of $\frac{2}{10}$. Each network has a width of 10 units in all layers, 3 layers, and data is synthetic regression data.



(a) 10 Hidden Units

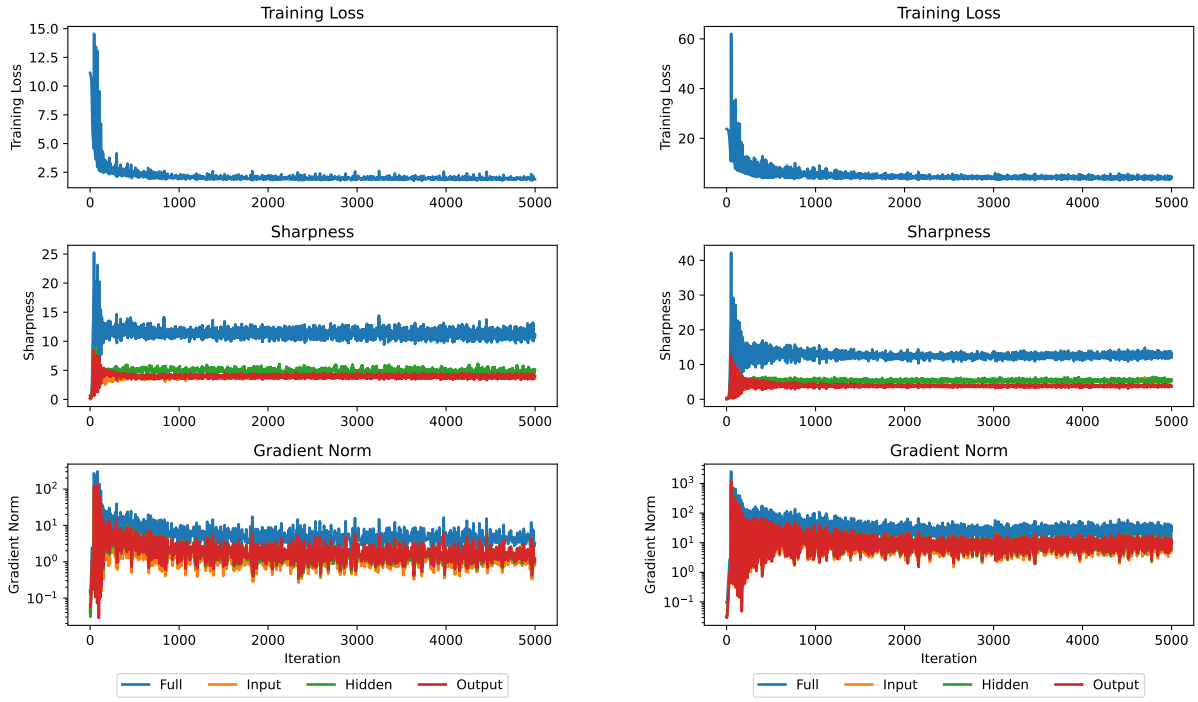
(b) 25 Hidden Units



(c) 50 Hidden Units

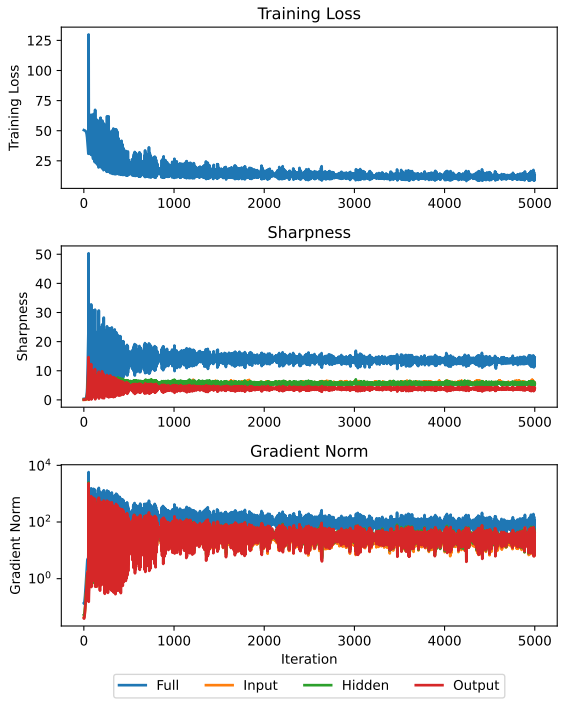
(d) 100 Hidden Units

Figure A.2: Edge of Stability compared across different widths with the tanh activation functions using a common stepsize of $\frac{2}{10}$. Each network has 3 layers, all layers have the same number of hidden units, and data is synthetic regression data.

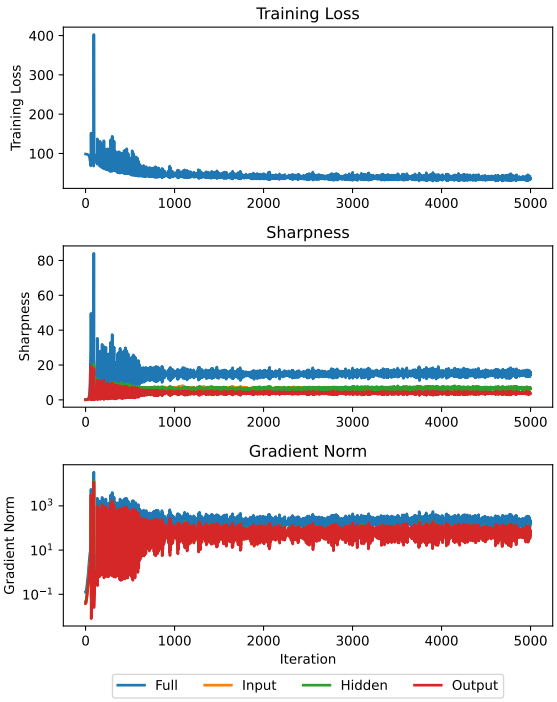


(a) 10 Hidden Units

(b) 25 Hidden Units

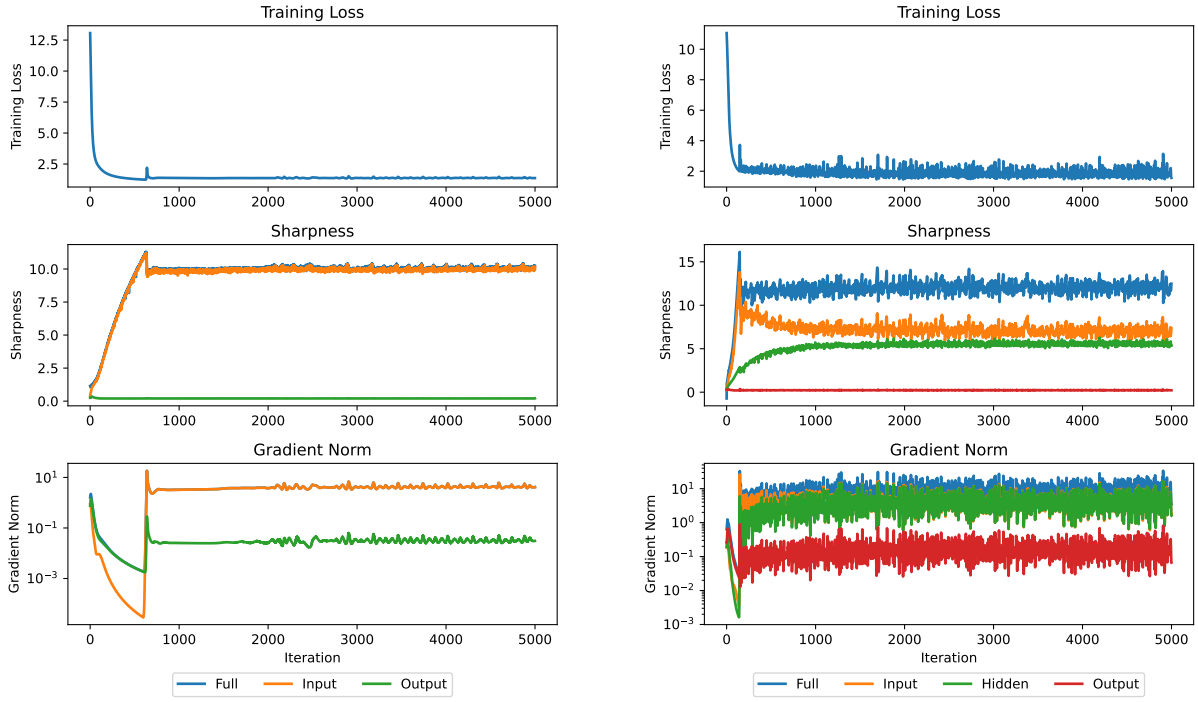


(c) 50 Hidden Units



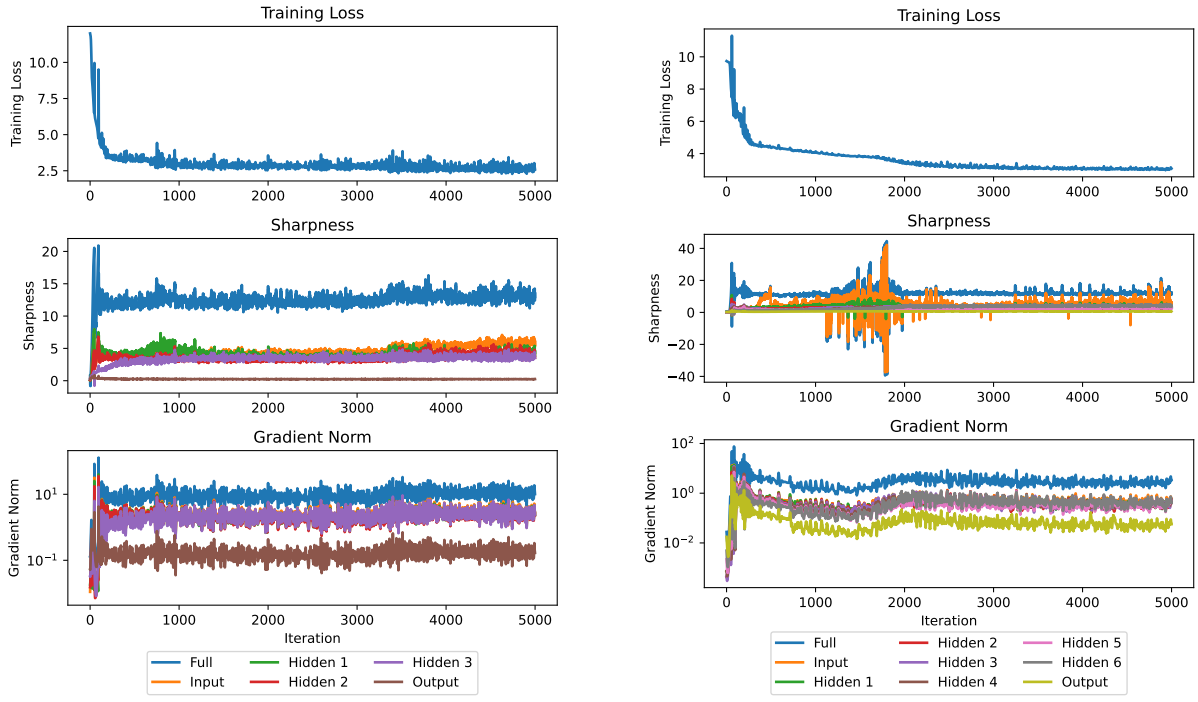
(d) 100 Hidden Units

Figure A.3: Edge of Stability compared across different widths with the ReLU activation functions using a common stepsize of $\frac{2}{10}$. Each network has 3 layers, all layers have the same number of hidden units, and data is synthetic regression data.



(a) 2 layers

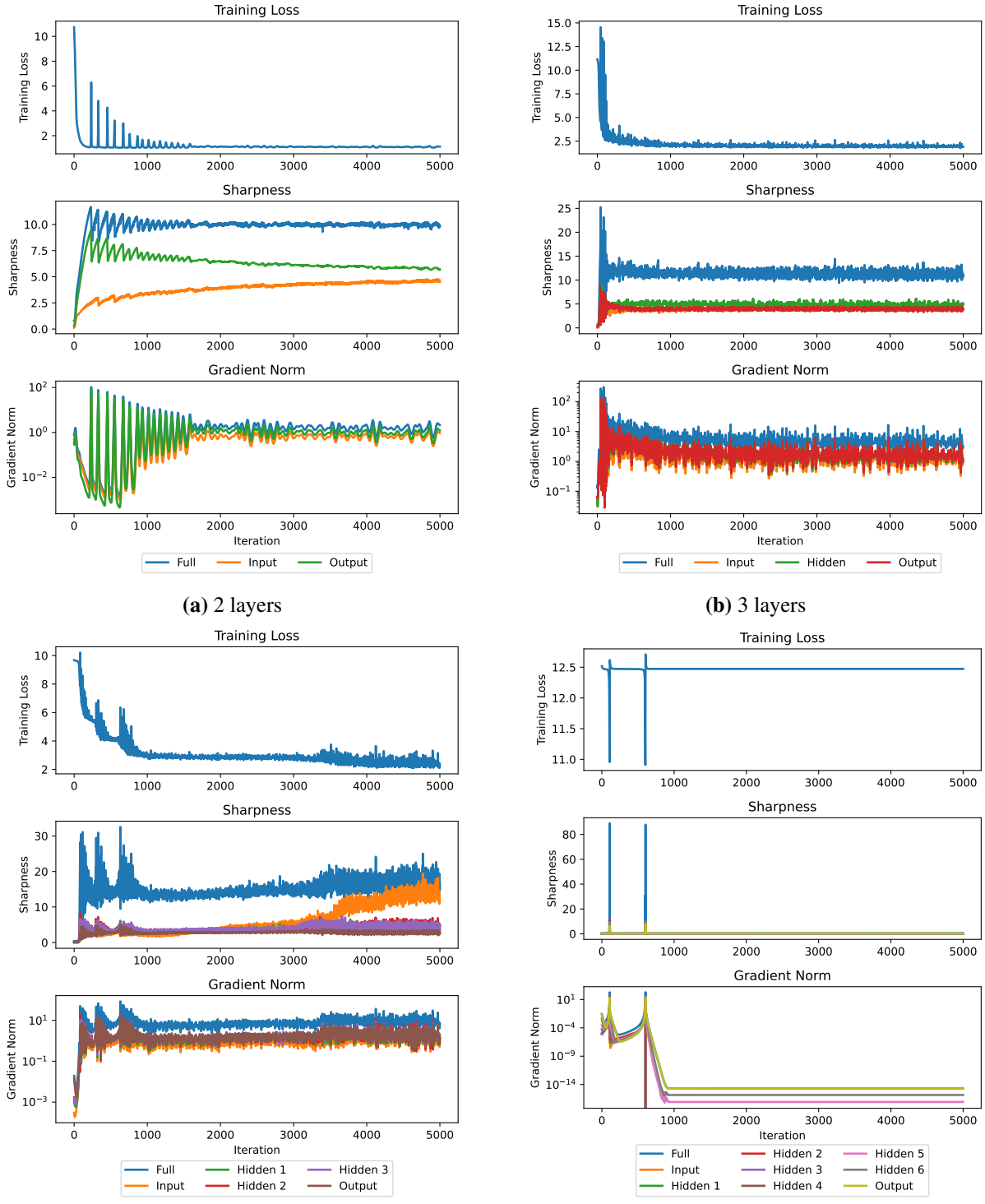
(b) 3 layers



(c) 5 layers

(d) 8 layers

Figure A.4: Edge of Stability compared across a varying number of layers with tanh activation functions using a common stepsize of $\frac{2}{10}$. All networks trained have 10 hidden units in all layers, and data is synthetic regression data.



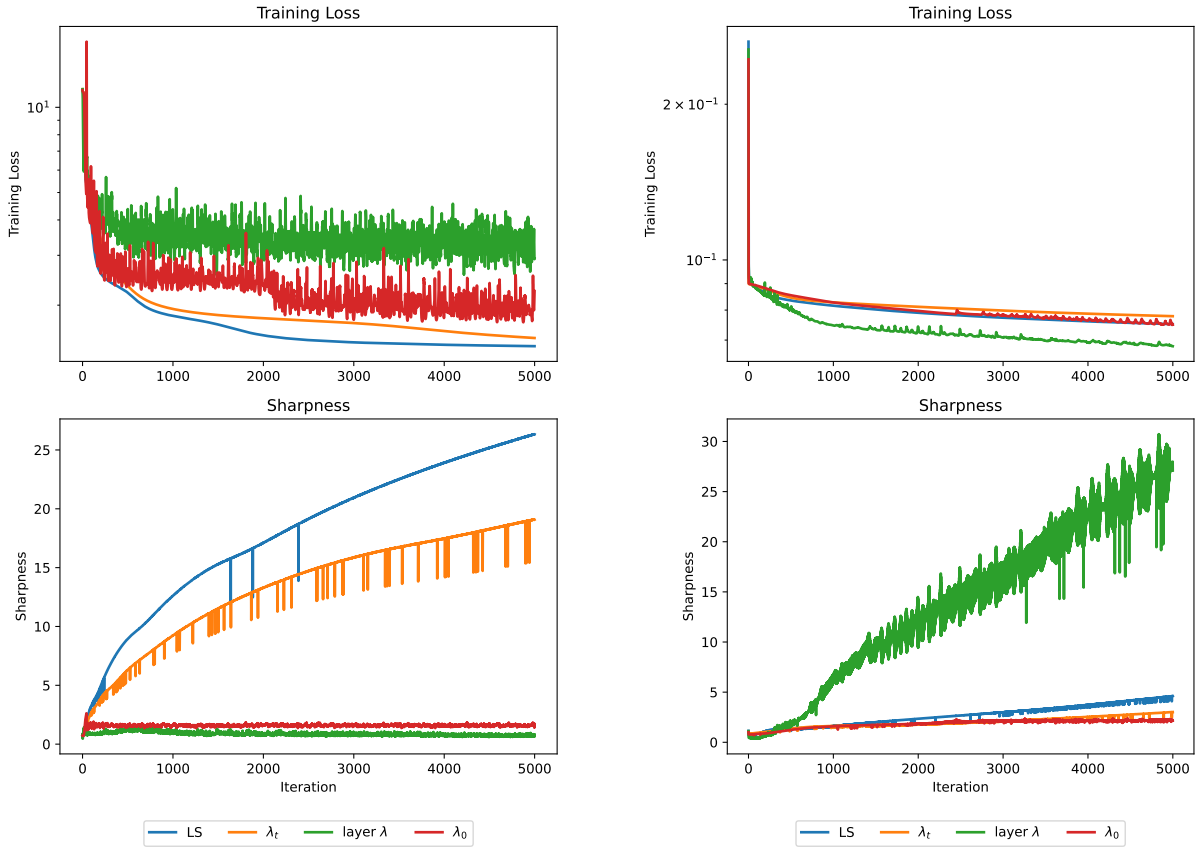
(a) 2 layers

(b) 3 layers

(c) 5 layers

(d) 8 layers

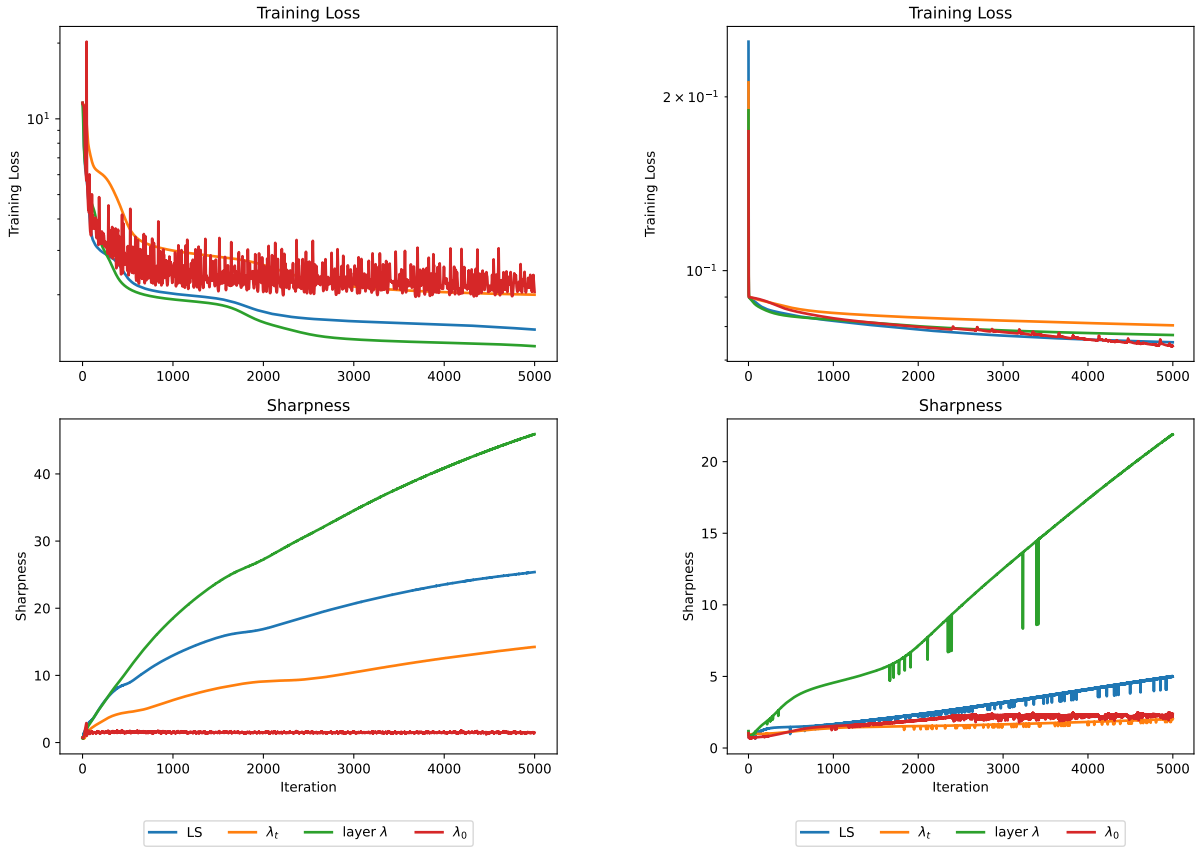
Figure A.5: Edge of Stability compared across a varying number of layers with ReLU activation functions using a common stepsize of $\frac{2}{10}$. All networks trained have 10 hidden units in all layers, and data is synthetic regression data.



(a) Synthetic Data

(b) CIFAR-10

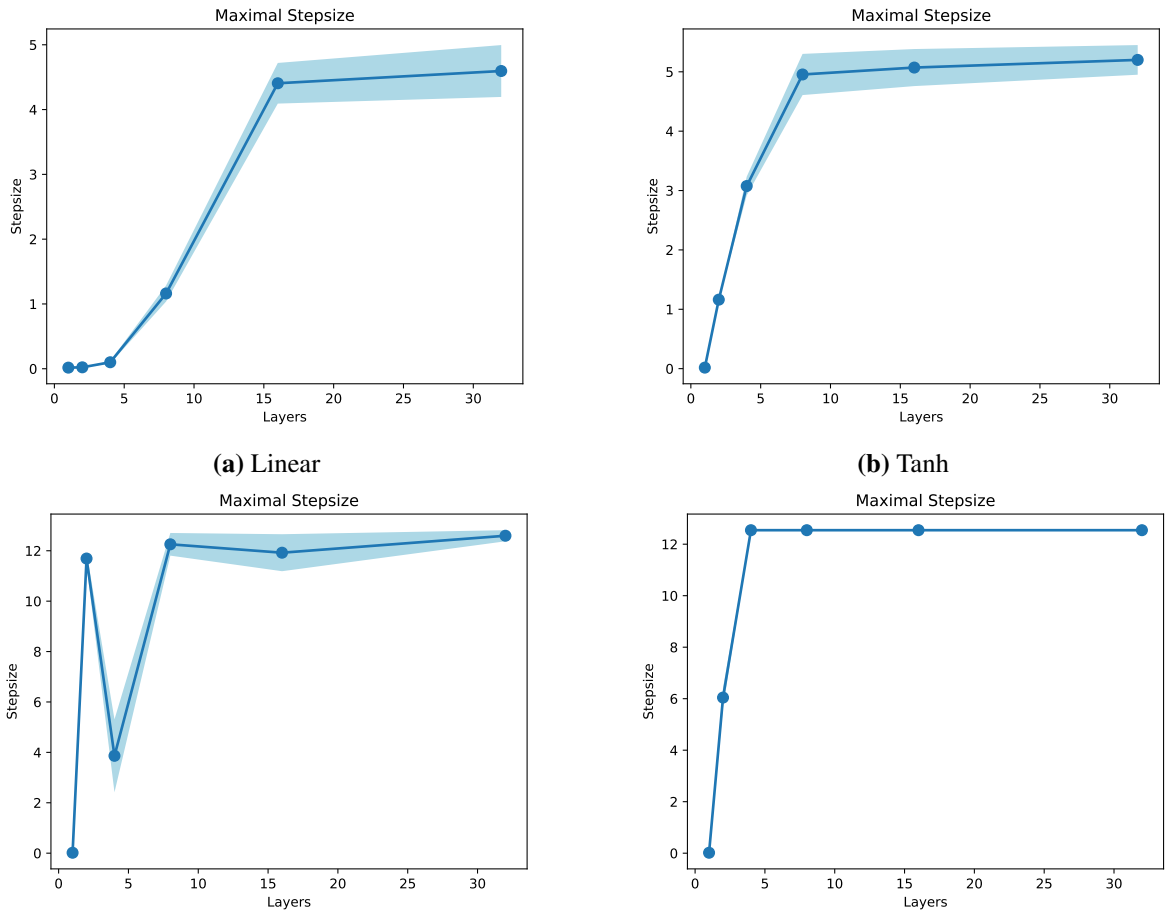
Figure A.6: Convergence for different step sizes and datasets, with sigmoid activations.



(a) Synthetic Data

(b) CIFAR-10

Figure A.7: Convergence for different step sizes (scaled) and datasets, with sigmoid activations.



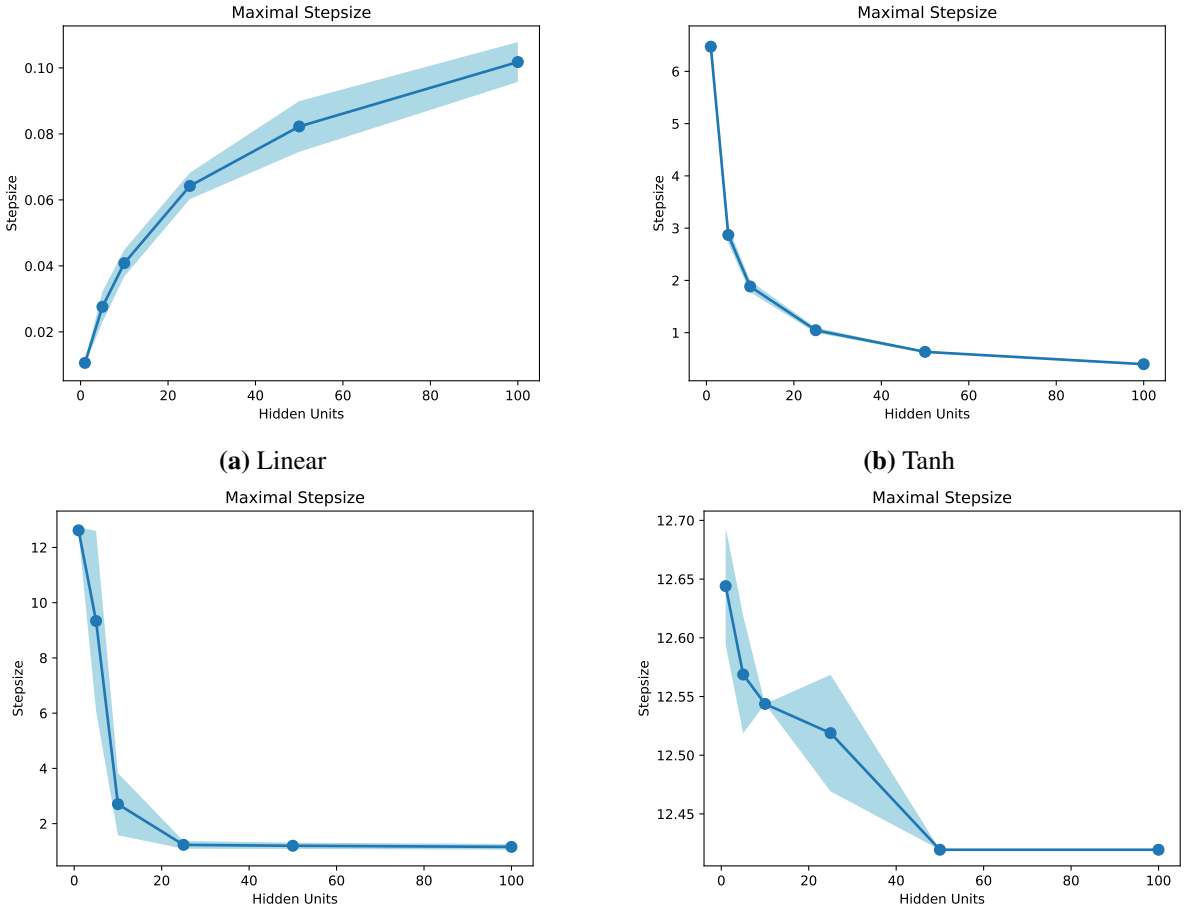
(a) Linear

(b) Tanh

(c) ReLU

(d) Sigmoid

Figure A.8: Compare the maximum step size that can be selected before gradient descent diverges across a number of depths for different activation functions, with each network having 10 hidden units per layer and 100 training examples used for training. Note that the dataset used is CIFAR10.



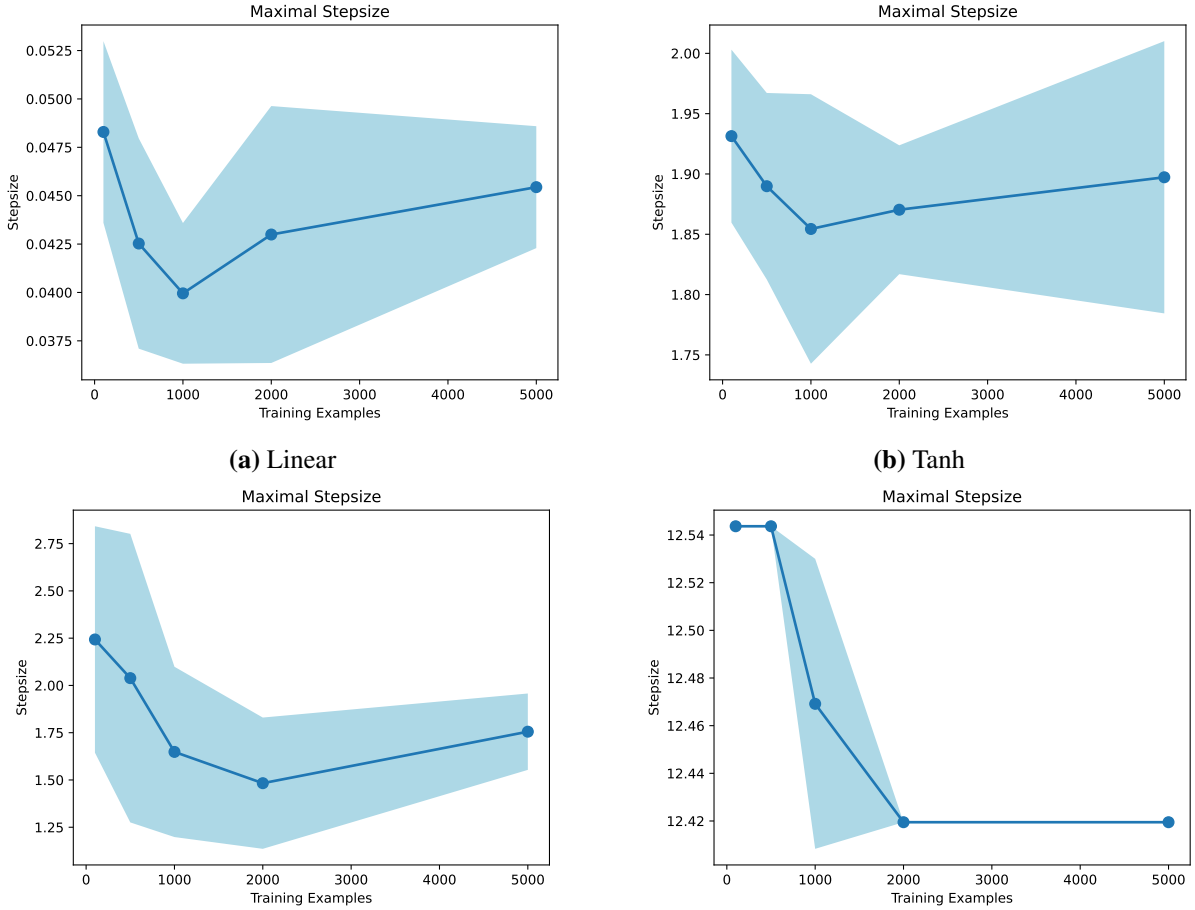
(a) Linear

(b) Tanh

(c) ReLU

(d) Sigmoid

Figure A.9: Compare the maximum step size that can be selected before gradient descent diverges across a differing number of hidden units per layer for different activation functions, with each network having 3 layer and 100 training examples used for training. Note that the dataset used is CIFAR10.



(a) Linear

(b) Tanh

(c) ReLU

(d) Sigmoid

Figure A.10: Compare the maximum step size that can be selected before gradient descent diverges across a varying number of training examples for different activation functions, with each network having 3 layers and 10 hidden units per layer. Note that the dataset used is CIFAR10.