Math 273A Final Project - Optimization of Linear Neural Networks

James Chen, Marc Andrew Choi, Marco Scialanga

Abstract

Neural Networks are a very popular subject of modern research. Our project focuses primarily on the convergence and (square) loss landscape of gradient descent for Linear Neural Networks.

1 Introduction

With the amazing achievements of neural networks today, one would expect a solid theoretical backing to the powerful empirical phenomenon. Yet, the gap between what is algorithmically state-of-the-art and what is rigorously sound is larger than one should be comfortable with. Thankfully, research is rapidly catching up, and one reason for its alacrity is the linear neural network. By simplifying the model while preserving the key structure, researchers like Nguegnang et. al., Achour et. al., and Lu and Kawaguchi are able to deeply study crucial properties of the networks and motivate promising generalized results for their nonlinear counterparts.

1.1 Basic Definitions and Notation

A neural network is a function $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ that takes as input observations from a matrix $X \in \mathbb{R}^{d_x \times m}$ and outputs a matrix $Y \in \mathbb{R}^{d_y \times m}$ such that

$$f(x) = f_{W_1,\dots,W_N,b_1,\dots,b_N}(x) = g_N \circ g_{N-1} \circ \dots \circ g_1(x) \tag{1}$$

By naming conventions, let $d_x = d_0$, $d_y = d_N$, where d_i denotes the number of rows/columns of the ith parameter (weight) matrix W_i. We define layers $g_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ to be the compositions of an affine function and a componentwise function, i.e.,

$$g_i(z) = \sigma(W_i z + b_i) \text{ for } W_i \in W^{\mathbb{R}^{d_i} \times \mathbb{R}^{d_i}}, b_i \in \mathbb{R}^{d_i} \tag{2}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ applied to a vector is treated as $(\sigma(w))_k = \sigma(w_k)$. The function σ is commonly known as the activation function and prominent examples include $\sigma(t) = \text{ReLU}(t) = \max\{0, t\}$ and $\sigma(t) = \tanh(t)$. It is common to see each W_i being called weights and b_i being called biases. The purpose of bias is to "delay" the activation function which allows for more flexibility in the network. Linear neural networks are neural networks with activation functions consisting of identity functions and biases $b_i = 0 \; \forall i$. Neural networks adapt parameters $W_1 \dots W_N$ based on training data pairs $(x_i, y_i) \in \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ consists of input data x_i and output data y_i. The way this model learns is performed through optimization. Given a loss function $\ell : \mathbb{R}^{d_y} \to \mathbb{R}^{d_y} \to \mathbb{R}_+$, one hopes to optimize

$$\mathcal{L}(W_1, \dots W_N, b_1, \dots, b_N) = \sum_{i=1}^{m} \ell(f_{W_1,\dots W_N,b_1,\dots,b_N}(x_i, y_i)) \tag{3}$$

Unless stated otherwise, our loss function will be the Frobenius norm. We also denote $W := W_N \cdots W_1$, giving us the following optimization problem:

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

$$\mathcal{L}(W_1, \ldots, W_N) = ||Y - WX||_F^2, \tag{4}$$

We recall definitions about critical points. Let $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ be the function that maps $x \mapsto \mathcal{L}(x)$. We assume $f \in C^2$ and denote $\nabla\mathcal{L}$ and $\nabla^2\mathcal{L}$ is its gradient and Hessian respectively. We also write $A \succeq 0$ if matrix A is positive semi-definite.

1. x^* is a global minimizer if and only if for all $x \in \mathbb{R}^n$, $\mathcal{L}(x^*) \leq \mathcal{L}(x)$
2. x^* is a local minimizer if there exists $\varepsilon > 0$ such that $\mathcal{L}(x^*) \leq \mathcal{L}(x)$ for all $x \in \mathbb{R}^n$ and $||x - x^*|| < \varepsilon$
3. x^* is a first order critical point if and only if $\nabla\mathcal{L}(x^*) = 0$
4. x^* is a second order critical point if and only if $\nabla\mathcal{L}(x^*) = 0$ and $\nabla^2\mathcal{L}(x^*) \succeq 0$
5. x^* is a strict saddle point if it is a first-order critical point but not a local minimizer, local maximizer, nor a second-order critical point
6. x^* is a non-strict saddle point if it is a first-order and second-order critical point but not a local minimizer nor local maximizer

Ultimately, we hope that gradient descent algorithms converge to global minimizers. However, due to the non-convex nature of the problem, it is not always feasible and we instead might converge to local minimizers, strict saddle points, and non strict saddle points. In what follows, we give an overview of when we can expect local minimizers to be global minimizers, when gradient descent will indeed converge to a global minimizer, and how saddle points can influence the convergence of gradient descent.

1.2 Our Contribution

We ran numerical experiments on the theoretical claim that were proven by Nguegnang et. al. for convergence of fixed step-size gradient descent in Theorem 2.6 of "Convergence of Gradient Descent for Learning Linear Neural Networks", expanding on their own numerical experiments in section 5 of their paper.[7] We also numerically verified the intuition Achour et. al. have in section 1.4 of their paper "The Loss Landscape of Deep Linear Neural Networks: a Second-order Analysis":

> "We show that the non-strict saddle points are associated with rmax plateau values of the empirical risk, where rmax is the size of the thinnest layer of the network (see Theorem 1). Typically these are values of the empirical risk that first-order algorithms can take for some time, as in Figure 3, and which might be confused with a global minimum."[1]

1.3 Outline of the report

First, we adapt a proof of a local-to-global result about first-order critical points of the loss landscape from Lu and Kawaguchi.[6] Next, we reference the classification of the second-order critical points of the loss landscape of linear networks attained in Achour et. al. and discuss the interactions between these different types of second-order critical points and gradient descent.[1] Finally, we examine a fixed step size upper bound (from Nguegnang et. al.[7]) for a linear neural network's convergence when using gradient descent. For the latter 2 topics, we discuss the theory and provide numerical experiments to fortify our results. We conclude with discussion of a slightly pessimistic state of the field going into the future.

2 First Order Analysis of the Loss Landscape

For the following section, we adapt and refer to the result in Lu and Kawaguchi[6] .

2.1 An Equivalent Loss

For a given Y, define

$$L(\{W_i\}) := ||Y - XW_1 \cdots W_N||_F^2 \tag{5}$$

and let

$$m := \min_{W_i \in \mathbb{R}^{d_{i-1} \times d_i}} L(\{W_i\}) \tag{6}$$

and $\{\hat{W}_i\}$ be the minimizing weight matrices. Notice that $r := \min_i d_i$ restricts the rank of the product $\hat{W} := \hat{W}_1 \cdots \hat{W}_N$ of our weight matrices. Define

$$G(M) := ||Y - XM||_F^2 \tag{7}$$

and consider the minimization problem

$$\min_{M \in \mathbb{R}^{d_x \times d_y}} G(M) \qquad \text{s.t.} \qquad \text{rank}(M) \leq r \tag{8}$$

Since $\text{rank}(\hat{W}) \leq r$, we have that

$$m = L(\{\hat{W}_i\}) = G(\hat{W}) \geq \left(\min_{M \in \mathbb{R}^{d_x \times d_y}} G(M) \qquad \text{s.t.} \qquad \text{rank}(M) \leq r \right) \tag{9}$$

Now let

$$n = \min_{M \in \mathbb{R}^{d_x \times d_y}} G(M) \qquad \text{s.t.} \qquad \text{rank}(M) \leq r \tag{10}$$

and $\hat{M}$ be a minimizer. Take an SVD of $\hat{M}$. With proper modification, i.e.:

- Appending 0 vectors to the appropriate unitary matrix
- Removing 0 vectors from the diagonal matrix along with their corresponding singular vectors from the unitary matrix
- Multiplying the diagonal matrix with the left or right unitary matrix so as to fit the dimensions d_i (for the case of $N = 2$)
- Sandwiching appropriately sized $(d_{i-1} \times d_i)$ identity matrices between the unitary matrices and the diagonal matrix

we can decompose $\hat{M}$ through this intermediary SVD into (possibly many identity matrices):

$$\hat{M} = \hat{M}_1 \cdots \hat{M}_N \tag{11}$$

with the M_i all having the correct dimension so that

$$n = G(\hat{M}) = L(\{\hat{M}_i\}) \geq \min_{W_i \in \mathbb{R}^{d_{i-1} \times d_i}} L(\{W_i\}) = m \tag{12}$$

Thus, $n = m$ and 6 has the same (globally) minimizing value as 8. With this fact, we can move on to prove something about the local minima.

2.2 Equivalent Local Minimizers

First, the local minimizers of 6 are shown to correspond to inputs of G that attain the same value as a local minimizer of 8.

Theorem 2.1 of Lu and Kawaguchi[6] Assume that X and Y have full row rank. If $\{\bar{W}_i\}$ is a local minimum of L, then $\bar{R} = \bar{W}_N \cdots \bar{W}_1$ achieves the value of a local minimum of G.

A sketch of the proof with multiple lemmas assumed is as follows: there exists, by lemma, an equivalent local minimizer $\{\hat{W}_i\}$ of L where

$$L(\{\hat{W}_i\}) = L(\{\bar{W}_i\}) \qquad \text{s.t. the product} \qquad \hat{R} = \hat{W}_1 \cdots \hat{W}_N \tag{13}$$

has rank r. If R is a perturbation of $\hat{R}$ with rank$(R) \leq r$, then there exists, again by lemma, a decomposition of R

$$R = W_1 \cdots W_N \tag{14}$$

s.t. W_i is a perturbation of $\hat{W}_i$. Since $\{\hat{W}_i\}$ is a local minimizer,

$$G(R) = L(\{W_i\}) \geq L(\{\hat{W}_i\}) = G(\hat{R}) \tag{15}$$

This holds true for all perturbations R of $\hat{R}$, so $\hat{R}$ is a local minimizer. Finally, notice that

$$G(\hat{R}) = L(\{\hat{W}_i\}) = L(\{\bar{W}_i\}) = G(\bar{R}) \tag{16}$$

Thus, $\bar{R}$ indeed attains the value of a local minimum of G. We defer to Lu and Kawaguchi for further details.[6]

2.3 No Strictly Local Minimizers for Rank Restricted Problem

Next, the local minimizers of 8 are shown to, in fact, be global minimizers.

Theorem 2.2 of Lu and Kawaguchi[6] If X has full row rank, all local minima of optimization problem 8 are global minima.

The proof of this result doesn't require too much specialized knowledge, but it is long. We defer to Lu and Kawaguchi for a proof.[6]

2.4 Conclusion of First-Order Analysis

The proof concludes as such: assume $\{W_1, \ldots, W_N\}$ is a local minimizer of L. By Theorem 2.1 of Lu and Kawaguchi, $W = W_1 \cdots W_N$ attains the value of a local minimizer of G. But, by Theorem 2.2 of Lu and Kawaguchi, W must attain the global minimum of G, making $\{W_i\}$ attain the global minimum of L.[6]

3 Second Order Analysis of the Loss Landscape

In this section, we will present a brief summary of the theory developed in[1] to introduce the numerical experiments in section 4.

3.1 Motivation

Strict and non strict saddle points can affect the behavior of gradient descent algorithms when minimizing the loss function. As a result, a second order analysis of the loss landscape for linear networks is necessary to better understand convergence of minimization algorithms applied to this problem.

3.2 Saddle Points of the Loss Function for Linear Networks

As expressed in the introduction, we want to minimize the loss function

$$\mathcal{L}(W_1, \ldots, W_N) = ||Y - W_N \ldots W_1 X||_F^2, \tag{17}$$

where $X \in \mathbb{R}^{d_x \times m}$ is our input data and $Y \in \mathbb{R}^{d_y \times m}$ our output data. We also define the following matrices

$$\Sigma_{XX} = \sum_{i=1}^{m} x_i x_i^T = XX^T \in \mathbb{R}^{d_x \times d_x}$$

$$\Sigma_{XX} = \sum_{i=1}^{m} x_i x_i^T = XX^T \in \mathbb{R}^{d_x \times d_x}$$

$$\Sigma_{XY} = \sum_{i=1}^{m} x_i y_i^T = XY^T \in \mathbb{R}^{d_x \times d_y}$$

$$\Sigma_{YX} = \sum_{i=1}^{m} y_i x_i^T = YX^T \in \mathbb{R}^{d_y \times d_x}$$

Using these newly defined matrices, we will assume the following

1. $d_y \leq d_x \leq m$
2. Σ_{XX} is invertible
3. Σ_{XY} is full rank (rank is d_y)
4. $\Sigma^{1/2} = \Sigma_{YX}\Sigma_{XX}^{-1}X \in \mathbb{R}^{d_y \times m}$ and $\Sigma = \Sigma^{1/2}(\Sigma^{1/2})^T \in \mathbb{R}^{d_y \times d_y}$

We can perform singular value decomposition on $\Sigma^{1/2} = U\Delta V^T$ where U, V are unitary and Δ is a diagonal matrix of descending singular values. Let $W = (W_H, \ldots, W_1)$ be a first-order critical point of the loss function and set $r = \text{rank}(W_H \cdots W_1) \in [0, r_{\max}]$. There exists a unique subset S of indices in $[1, d_y]$ of size r such that

$$W_H \cdots W_1 = U_S U_S^T \Sigma_{YX} \Sigma_{XX}^{-1} \tag{18}$$

W is a critical point associated with S. We can find the exact value by

$$L(W) = \text{tr}(\Sigma_{YY}) - \sum_{i \in \S} \lambda_i \tag{19}$$

Now, to differentiate saddle points, recall:

Definition Let H be the hessian of our loss function. A saddle point W_s is strict if $H(W_s)$ has a negative eigenvalue. A saddle point W_{ns} is non strict if $H(W_{ns})$ is positive semidefinite with at least one eigenvalue equal to zero.

- We can think of non strict saddle points as flat valleys, while the loss at strict saddle points has negative curvature in at least one direction.
- For gradient descent type algorithms, it is much harder to escape non strict saddle points than strict ones.

The global minimizer W^* of the loss function is the projection of the linear regression solution $\hat{\beta} = (X'X)^{-1}X'Y$ onto a lower dimensional subspace generated by the first $rmax$ (minimum dimension of the network matrices) eigenvectors of the sample covariance matrix of $\hat{Y}$, which we denote as Σ.[3] Projecting $\hat{\beta}$ onto the subspace generated by other subsets of eigenvectors of Σ allows us to find saddle points.[1] Interestingly, for a linear network with 3 or more layers, every saddle point W_{crit} that is obtained by projecting $\hat{\beta}$ on the first $r < rmax$ eigenvectors of Σ can be either a **strict** or a **non strict** saddle point, depending on the parametrization.[1] Furthermore, these saddle points correspond to:

$$\underset{M \in \mathbb{R}^{d_x \times d_y}, rank(M) \leq r}{\arg\min} ||Y - MX||_F^2 \tag{20}$$

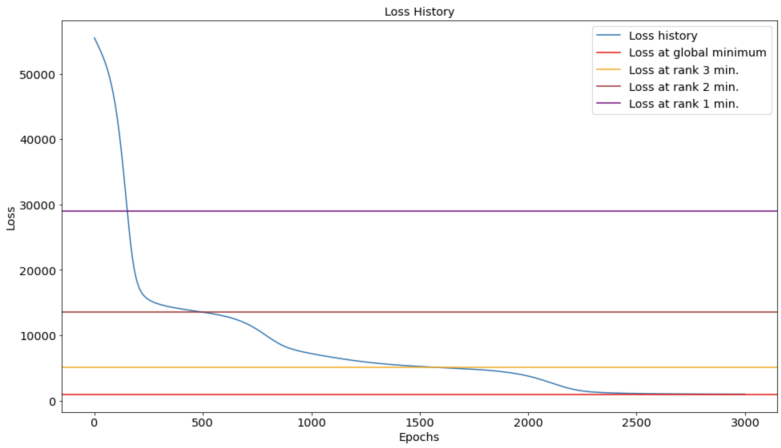

Figure 1: The loss plateaus in the vicinity of low rank minimizers.

4 Numerical Experiment on Critical Points

We attempted to corroborate this result in a multivariate linear regression setting. We initialize our data matrix $X \in \mathbb{R}^{1000\times 5}$ randomly from the standard normal distribution. We let and $Y = X\beta + \mathcal{E}$, where the predictor matrix $\beta = diag(5, 4, 3, 2, 1) \in \mathbb{R}^{5\times 5}$ and $\mathcal{E} \in \mathbb{R}^{1000\times 5}$ with each entry being drawn from $N(0, 0.1)$. We build a three layer linear network with $W_1 \in \mathbb{R}^{5\times 4}$, $W_2 \in \mathbb{R}^{4\times 4}$, $W_3 \in \mathbb{R}^{4\times 5}$. This leads our model to have $rmax = 4$, because we have a 4×4 matrix. Our loss function then looks like:

$$\mathcal{L}(W_1, W_2, W_3) = ||Y - XW_1W_2W_3||_F^2. \tag{21}$$

Then, after taking the appropriate transposes to match with the definitions given above, we compute $\Sigma_{YX}, \Sigma_{XX}, \Sigma_{YY}, \Sigma^{1/2}$. We can quickly check that the assumptions above are all satisfied. This is due to the randomness of the data. We can then find the global minimizer W^* using Theorem 1 in:[1]

$$W^* = U_S U_S^T \Sigma_{YX} \Sigma_{XX}^{-1},\ S = [1, 2, 3, 4]. \tag{22}$$

This critical point corresponds to a matrix very close (because of the random error) $\mathcal{E}$ to $diag(5, 4, 3, 2)$, as expected. Our neural network does indeed converge to this point, and the loss can be computed as $\mathcal{L}(W^*) = tr(\Sigma_{YY}) - \sum \lambda_i$, where λ_i are the singular values of Σ as defined in Section 2.1.

Furthermore, we can find other critical points by taking different subsets of $[1, 4]$ as our indexes for U, as the theorem states. To investigate the phenomenon of implicit regularization, we are particularly interested in the low rank minimizers, so we find them and compute the loss at these points.

In 1 we can see the loss history when training our network with random weight initialization. We notice that before converging to the global minimum, gradient descent seems to sequentially solve the following constrained minimization problem:

$$\underset{M\in\mathbb{R}^{d_x\times d_y}, rank(M)\leq r}{\arg\min} ||Y - XM||_F^2 \tag{23}$$

This example of implicit regularization was mentioned in[1] and it is a current research topic. To explore this concept further, we parametrize the rank 3 minimizer as a strict and non strict minimizer. We notice that for this network, to escape a non strict saddle point it took around 2000 epochs, while it only took about 500 iterations to leave a strict saddle point. Since the loss with random initialization (in green) seems to be plateau for around 500 epochs as well, we hypothesize that gradient descent, during the process of sequentially finding low rank minimizers, avoids non strict saddle points in favor of strict ones.

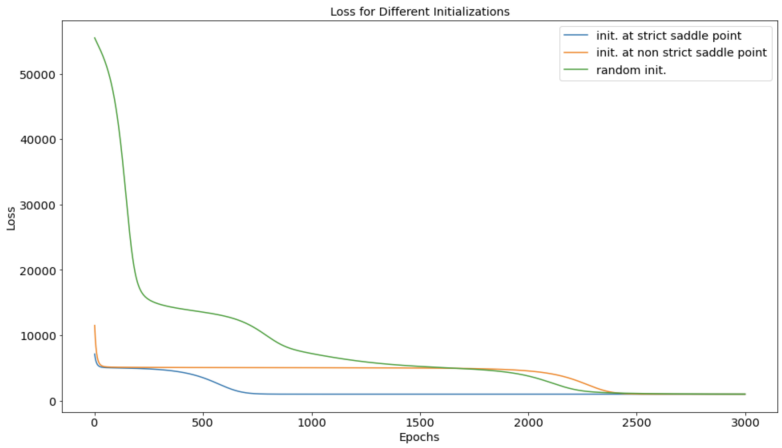

Figure 2: Loss with random initialization vs initialization at rank 3 minimizer (parametrized as strict and non strict saddle point).

5 Fixed Step Size Convergence

5.1 Introduction and Motivation

In class, we saw observed that for the quadratic function $f(x) = \frac{1}{2}x^TQx - b^Tx$ where $Q \in \mathbb{R}^{n\times n}$ symmetric positive definite and $b, x \in \mathbb{R}^n$, the fixed step sized gradient descent algorithm converges for any initialization if and only if $0 < \alpha < \frac{2}{\lambda_{\max}(Q)}$.[4] We aim to find a similar bound for a fixed step size gradient descent algorithm applied to a linear neural network. The main theorem we will prove is presented in Nguegnang's paper[7] but we will go over the main components since it provides educational value.

5.2 Gradient Flow

Since gradient descent is applied to discretized data, we hope to generalize the algorithm in a more general sense, e.g. the continuous setting. To accomplish this, we look to gradient flow. In a holistic view, gradient flow is a system of differential equations representing gradient descent in a continuous case where we can use principles in dynamical systems to understand the behavior of the underlying structure. In our case, the gradient flow for the function loss function given weights $W(t) = (W_N(t)\cdots W_1(t))$ where $t \in \mathbb{R}^{\geq 0}$ is given by

$$\frac{dL}{dW_j} = -\nabla_{W_j} L(W_1(t), \ldots, W_N(t)), \quad j = 1, \ldots, N \tag{24}$$

with some initial weights $W(0) = (W_1(0), \ldots, W_N(0))$. Explicitly, the gradient of the loss function with respect to some weight matrix is given by[2]

$$\nabla_{W_j} L(W_1(t), \ldots, W_N(t)) = W_j XX^T - YX^T, \quad j = 1, \ldots, N \tag{25}$$

Using the two previous results, we can get to the next "time step" with the following assignment

$$W_j(t+1) = W_j(t) - \eta \frac{dL}{dW_j}(W_1(t), \ldots, W_N(t)), \quad j = 1, \ldots, N \tag{26}$$

To discuss gradient flow in more detail we provide an important definition. Weight matrices $W_1, \ldots, W_N$ are balanced if

$$W_{j+1}^T W_{j+1} = W_j W_j^T, \quad j = 1, \ldots, N-1 \tag{27}$$

We say the flow is balanced if the initial weights are balanced. This is a difficult condition to satisfy so we can relax the condition using balancedness constants. A tuple of weights have balancedness constant $\delta \geq 0$ if

$$||W_{j+1}^T W_{j+1} - W_j W_j^T|| \leq \delta, \quad j = 1, \ldots, N-1 \tag{28}$$

We can consider balanced weights to have balancedness constant 0. Now, we can find some bounds on $||W||$ which will ultimately help derive an upper bound for the fixed step size gradient descent algorithm applied to a linear neural network.

Proposition (Proposition 3.2 of Nguegnang et. al.[7]) Let W be a tuple of weights with balancedness constant $\delta \geq 0$. Then,

$$||W_j||^2 \leq ||W||^{\frac{2}{N}} + (N+1)\delta \tag{29}$$

We omit the proof of this proposition since it is given in[7] focus on other aspects of the paper such as numerical experiments.

5.3 Convergence Bound

Theorem 2.4 of Nguegnang et. al.[7] Let $X \in \mathbb{R}^{d_x \times m}$, $Y \in \mathbb{R}^{d_y \times m}$ be data matrices such that XX^T is full rank. Suppose the initial weights of the gradient descent algorithm has balancedness constant $\alpha\delta$ for some $\delta > 0$ and $\alpha \in [0, 1)$. Assume that the stepsizes $\eta > 0$ are constant and

$$\begin{aligned}
\eta &\leq \frac{2(1-\alpha)\delta}{4\mathcal{L}(W(0)) + (1-\alpha)\delta B_\delta} \text{ where} \\
B_\delta &= 2eNK_\delta^{N-1}||X||^2 + \sqrt{e}NK_\delta^{\frac{N}{2}-1}||XY^T|| \\
K_\delta &= M^{\frac{2}{N}} + (N+1)^2\delta \\
M &= \frac{\sqrt{2||Y - W_N(0)\cdots W_1(0)X|| + ||Y||}}{\sigma_{\min}(X)}
\end{aligned}$$

Then fixed step-size η gradient descent converges to the global minimum for almost all initializations of the linear neural network. Although this upper bound may seem rather strange and difficult to work with, with specific choices of δ, some of these formulas simplify nicely. We will see an example of a nice choice of δ in the following section.

5.4 Numerical Experiment

We create a numerical experiment to confirm the previous result. Let $X \in \mathbb{R}^{5\times 5}$ be a random matrix with entries sampled from a normal distribution. We have a three layer neural network with each weight $W_i \in \mathbb{R}^{5\times 5}$ sampled from a normal distribution with the layers having a balanceness constant of

$$\delta = \frac{1}{N(N+1)^2} M^{\frac{2}{N}} \qquad \text{where } M \text{ is defined as} \tag{30}$$

$$M = \frac{\sqrt{2||Y - W_N(0)\cdots W_1(0)X|| + ||Y||}}{\sigma_{\min}(X)} \tag{31}$$

and N is the number of layers. Our inspiration for our choice of δ comes from Ngeugnang's paper.[7] We let $Y = \text{diag}(1, 2, 3, 4, 5)X + \mathcal{E}$ where $\mathcal{E}$ is some noise which is randomly sampled from a Gaussian distribution with mean 0 and standard deviation of 0.1. With these definitions, we aim to minimize the loss function.

$$\mathcal{L}(W_1, W_2, W_3) = ||Y - XW_1W_2W_3||_F^2 \tag{32}$$

With our initial weight matrices, we have that $\delta = 1.17328$ and the balancedness constant is $\alpha\delta = 0.84352$. This gives us $\alpha = 0.71894$. We also checked and found that XX^T is full rank. Since we have satisfied all the conditions, we have a theoretical stepsize upper bound of $\eta = 2.374 \cdot 10^{-4}$. Since this is a sufficient condition and not a necessary and sufficient condition, it could be the case where some step size greater than η allows us to converge to a critical point. To demonstrate this, we started with 0.9η to ensure we indeed have convergence. Using 5000 epochs (iterations of the gradient descent algorithm), if a step size converges to the critical point, we double the step size. According to our figure 3, we see that with all these step sizes, we have convergence to the critical point. As we expect, with the smaller step size, we see convergence significantly slower compared to the larger step sizes. This implies that although we have a possible step size bound for convergence, this bound can be significantly tighter which could be possibly discussed in future works. Because we double the step size every time the method converges, the first step size in which we fail to converge to a critical point is $\eta = 0.0273$.

We also varied the parameters of the numerical experiment to see if perturbing other aspects of the model would result in different results. First, instead of having all the weights being 5×5 matrices,

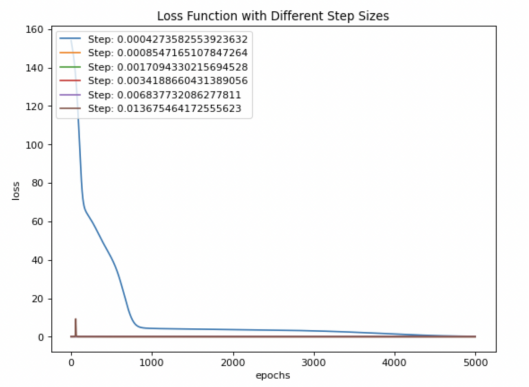

Figure 3: Convergence of linear neural networks with varying step sizes

we let $W_1(0) \in \mathbb{R}^{5\times 2}$, $W_2(0) \in \mathbb{R}^{2\times 2}$ and $W_3(0) \in \mathbb{R}^{2\times 5}$. To see the effect this has on the bound of convergence, we run the calculations with 1000 linear neural networks and graph the distributions.

Looking at 4, we see that the distribution of step sizes are roughly the same. Since the dimensions of the weights do not provide any insightful information of the step size bound, we investigate what occurs if we increase the number of weights in our linear neural network. As a result, we preform the same experiments but for a 4 layer neural network and a 2 layer neural network.

Looking at Figures 4 and 5, the convergence bound changes by about a factor of 10 as we increase the number of weights. For instance, a linear neural network with 2 weights has a step size convergence bound of about $\eta = 10^{-3}$, a linear neural network with 3 weights has a step size convergence bound of about $\eta = 10^{-4}$ and a linear neural network with 4 weights has a step size convergence bound of about $\eta = 10^{-4}$. This is roughly what we expect since the step size bound given changes exponentially with N.

In the original numerical experiment, we saw that we had to increase the step size significantly in order to force divergence. We wanted to see when this occurs in a general sense. We ran two different experiments to confirm what we believed. First, we ran the first experiment 100 times and created a histogram of step sizes. Next, we ran the second experiment (where all the weights were not 5×5 matrices) 100 times and created a histogram of step sizes. We only run the experiments 100 times rather than 1000 because of lack of computing power. The two plots are given in Figure 6.

When looking at the distribution of step size for divergence in the scenario with all 5×5 weights, we see that the distribution is skewed right with most of the linear neural networks requiring step sizes from $\eta = 0.01$ to $\eta = 0.03$ to diverge. However, if we look at the situation where we use

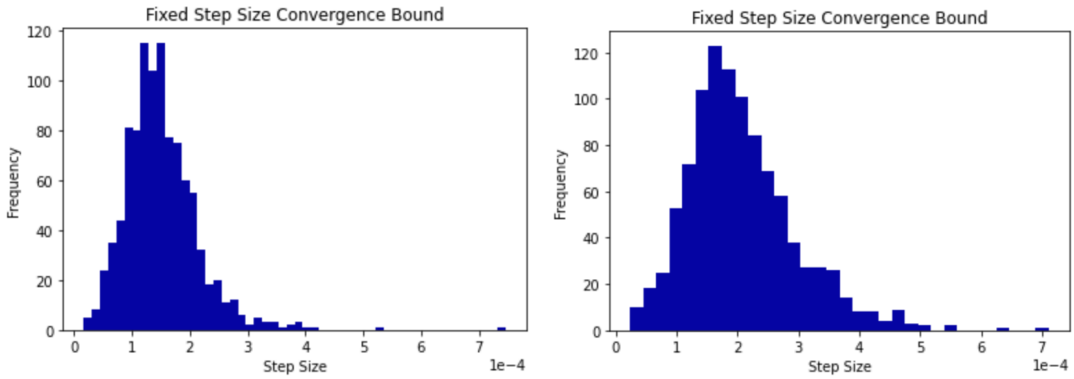

Figure 4: Left-Calculated Bound of Convergence of Linear Neural Networks using 3 5x5 weights. Right-Calculated Bound of Convergence of Linear Neural Networks using $W_1(0) \in \mathbb{R}^{5\times 2}$, $W_2(0) \in \mathbb{R}^{2\times 2}$ and $W_3(0) \in \mathbb{R}^{2\times 5}$ as initial weights

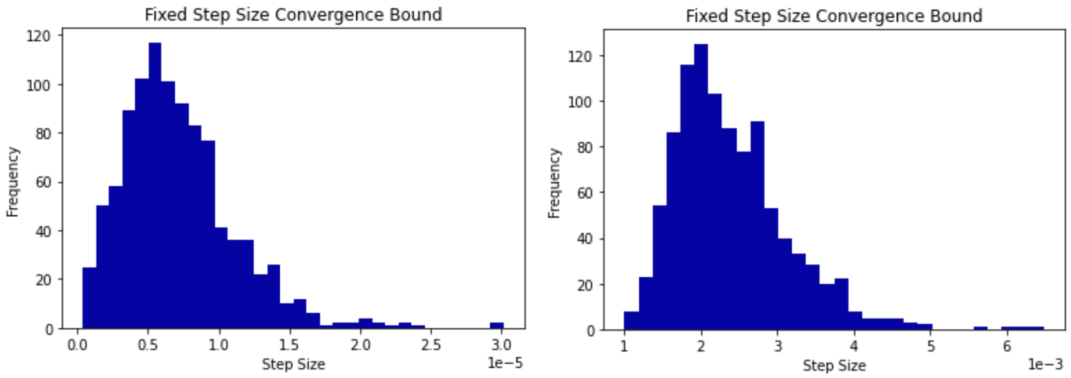

Figure 5: Left-Calculated Bound of Convergence of Linear Neural Networks with four layers. Right-Calculated Bound of Convergence of Linear Neural Networks with two layers

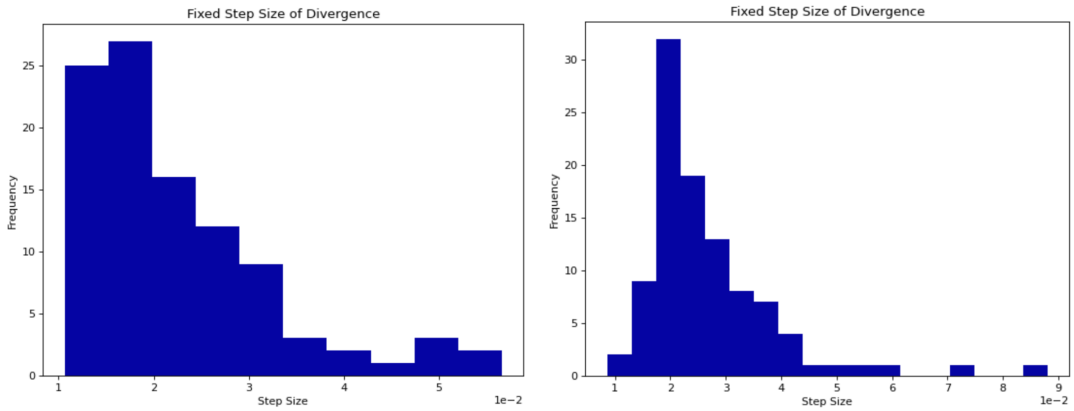

Figure 6: Left-Step size of Divergence of Linear Neural Networks using 5x5 weights. Right-Step size of Divergence of Linear Neural Networks using variable weights

$W_1(0) \in \mathbb{R}^{5\times 2}$, $W_2(0) \in \mathbb{R}^{2\times 2}$ and $W_3(0) \in \mathbb{R}^{2\times 5}$ as initial weights, we see that there is less variability and most neural networks tend to diverge upon having a step size of $\eta = 0.02$. Also, we have more possibility of outliers with larger step sizes of divergence in the latter case. For future works, instead of looking at an upper bound for convergence, it might be interesting to see a lower bound for divergence for linear neural networks.

6 Conclusion

With these results in place, we note some sequels on *non-linear* neural networks inspired by the character of these results on linear neural networks: Nguyen and Hein prove in their paper "The loss surface of deep and wide neural networks[8]" a corresponding result true (in a sense) for *wide non-linear* networks (under certain assumptions). Specifically, they prove that *almost every* local minimizer is a global minimizer (for their very specific network architectures). Hope for a full result, however, is quickly dashed upon further reading. In Their paper "On the Stability Properties and the Optimization Landscape of Training Problems with Squared Loss for Neural Networks and General Nonlinear Conic Approximation Schemes", Christof proves that the emergence of so-called "spurious" (not globally optimal) minima in the loss landscape is directly linked (in a quantifiable way, no less) to the increase in expressiveness that certain types non-linearities (specifically conic ones) bestow upon the neural network approximators.[5] All of this implies (for the specific cases considered) that the set of spurious local minima for losses of non-linear neural networks may have measure 0, but it will never be empty. This justifies the efforts that researchers have been putting

into figuring out how to overcome these local pitfalls, such as using stochastic methods to jiggle the optimization path out of these spurious local minima to achieve a more optimal point of convergence.

7 Members Contributions

This project was completed by James Chen, Marc Andrew Choi and Marco Scialanga. James touched up the introduction and fixed step size convergence as well as worked on the first-order analysis and conclusion/future works sections. Marc Andrew worked on the bound for convergence of fixed step sized gradient descent method. In particular, he wrote most of the code, created numerical experiments and completed the final report for those sections. Marco mainly worked on the second order analysis of the loss landscape, performing numerical experiments related to the paper "The loss landscape of deep linear neural networks: a second-order analysis" by Achour, Malgouyres, Gerchinovitz. Marco also wrote part of the introduction on linear networks and motivation for studying them.

References

[1] El Mehdi Achour, François Malgouyres, and Sébastien Gerchinovitz. The loss landscape of deep linear neural networks: a second-order analysis, 2021.

[2] Bubacarr Bah, Holger Rauhut, Ulrich Terstiege, and Michael Westdickenberg. Learning deep linear neural networks: Riemannian gradient flows and convergence to global minimizers, 2019.

[3] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[4] Edwin K. P. Chong and H. Zak, Stanislaw. *An introduction to optimization*. John Wiley and Sons, New Delhi, 2 nd edition, c2010.

[5] Constantin Christof. On the stability properties and the optimization landscape of training problems with squared loss for neural networks and general nonlinear conic approximation schemes, 2020.

[6] Haihao Lu and Kenji Kawaguchi. Depth creates no bad local minima. *arXiv preprint arXiv:1702.08580*, 2017.

[7] Gabin Maxime Nguegnang, Holger Rauhut, and Ulrich Terstiege. Convergence of gradient descent for learning linear neural networks, 2021.

[8] Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks, 2017.