# Experiments on the Generalization and Learning Dynamics of Deep Neural Networks

by

Robert Xinyu Liang

S.B., Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2019



Author .....
Department of Electrical Engineering and Computer Science
May 17, 2019

Certified by .....
Tomaso Poggio
Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

# Experiments on the Generalization and Learning Dynamics of Deep Neural Networks

by

Robert Xinyu Liang

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2019, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

## Abstract

In the recent few years, deep learning has had great successes in many applications such as image recognition. However, theory seems to lag behind application in this field, and one goal has been to provide principles and solve puzzles. Some goals during this thesis work were to develop new software tools for deep learning researchers, run experiments related to the research of CBMM (Center for Brains, Minds, and Machines), and create graphs for papers published by CBMM.

Thesis Supervisor: Tomaso Poggio
Title: Professor

# Acknowledgments

This work is partially funded by the NSF through the Center for Brains, Minds, and Machines.

I would like to express my very great appreciation to Professor Poggio, for not only being my supervisor, but also my mentor, and for his helpful attention.

I am grateful to my family and friends, for their support and love.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

# Chapter 2

# Deep Learning Framework

## 2.1 Introduction

At the start of my thesis work, I mainly focused on the development of a new deep learning framework. At the time of writing, this framework was still being worked on and has not yet been named. The primary goal of the framework was to create a Python library which could facilitate machine learning researchers by simplifying the process of making custom neural networks beyond what is usually supported by popular frameworks such as Pytorch, Tensorflow, and Keras. This involved thinking about and developing new coding patterns which could make coding neural networks more intuitive, all while enabling methods of training other than backpropagation.

## 2.2 Description

The central idea of our framework was that each fundamental computational unit of a neural network would be a function. Also, all of the data, whether it would be the input or the gradient calculated from a loss function, would be wrapped in a class that signifies the data type which would then tell a function which operation to perform. Each function would look at its input, detect whether it's a gradient, training example, or testing example, and use that to perform either a gradient update, training with techniques such as batch normalization enabled, or inference. The benefit of using

functions is the syntax can be easier to understand, and it is simple to compose multiple layers together into a group, such as a unit in a ResNet, and then compose those groups to form a larger network.

```
def net(in):
    mode = mode_of(in)
    if mode == "Training":
        return fc(relu(conv(relu(conv(in)))))
    elif mode == "Backprop":
        return conv(relu(conv(relu(fc(in)))))
```

Furthermore, because each function is self-contained, we brainstormed ways to share these layers and networks in a simple way through GitHub and also a website designed for researchers using this framework. In general, this framework would be helpful for CBMM's research, because one goal of the group is to look at various ways of artificial learning.

## 2.3 Challenges

After briefly working on this project at the start of my thesis work, I decided to work on a new research topic for a few reasons. One reason was I anticipated some of the challenges that are still faced by the students working on this project. The main challenge is gaining popularity in a research environment dominated by Pytorch and Tensorflow users. Other than that, some challenges are getting all of the pieces to work smoothly with a very small dev team. Tensorflow and Pytorch each have a huge team of engineers to maintain its code, add features, and respond to bugs.

Another reason was that a new project opened up in my lab. My work on this new project will be described in the following chapters, but the general idea is that for some of the research projects CBMM was working on, there were theoretical predictions that needed to be confirmed through experiments, while there were other theoretical hypotheses to be tested on new neural architectures such as ResNets.

# Chapter 3

# Experiments Involving Loss

## 3.1 Introduction

The first experiments I ran were on the effect of perturbations during training on the loss. This has the effect of studying the shape of the landscape of the loss around the minimum. By perturbing the weights we force training to reset slightly. While this has no expected positive impact on the final testing error, we do this to look at whether it can have a negative impact, and also how the initialization affects this negative impact. We are also interested in seeing the similarity between the solutions we converge on as we perturb. Because we were studying datasets that were linearly separable, I first needed a way to generate a dataset.

## 3.2 Data Generation

To produce a training set that is linearly separable, I generated data in $N$ dimensions sampled from the multivariate uniform random distribution. Then I generated a random perceptron classifier passing through the origin, and used this to label the randomly generated data. This was accessed through a function that could produce training and testing data of any desired size and from the same distribution.

## 3.3 Network Configuration

I started these experiments with a simple neural network with an $N$ dimensional input, no hidden layer, and 2 outputs. In other words, the first output is equal to

$$\hat{y}^{(1)} = \mathbf{W}_1^T \mathbf{x} + b_1$$

and the second output is equal to

$$\hat{y}^{(2)} = \mathbf{W}_2^T \mathbf{x} + b_2$$

where $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{x}$ are $N$-dimensional vectors.

The loss function used is the cross-entropy loss on two classes:

$$L(\hat{\mathbf{y}}) = -\mathbf{y} \cdot \log \hat{\mathbf{y}}$$

## 3.4 Effect of Weight Initialization on Loss

I studied the effect of varying weight initialization on training and testing loss. I varied weight initialization by selecting initial weights and biases from a normal distribution with varying standard deviations: 0.01, 0.05, and 0.1. Then I trained the network using stochastic gradient descent (SGD) with 10 passes through the data. The resulting training and testing loss is plotted in Figures 2-1 to 2-3. As you can see, with a larger standard deviation of initial weights and biases, it takes slightly longer for the network to converge. In the loss graphs, one feature of note is that the loss oscillates slightly after recovering from a perturbation. However, given enough time, the loss converges back to the baseline.

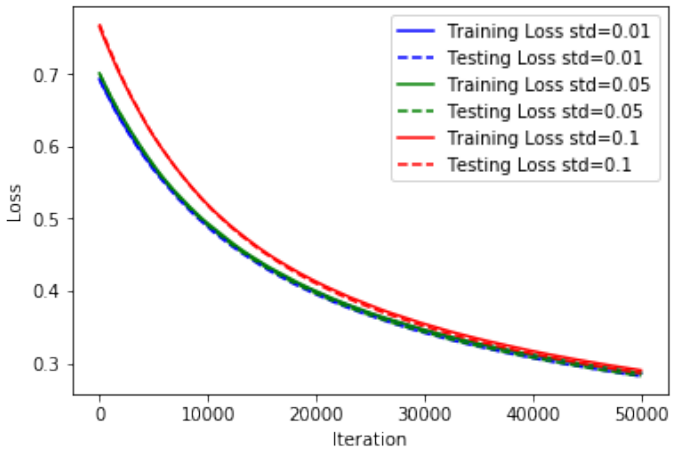



Figure 3-1: Loss over 10 epochs, learning rate = 0.0001, produced by a network with no hidden layers, with 5 inputs, 2 outputs, and cross-entropy loss on linearly separable data. Used SGD with batch size of 1 and shuffled input data of size 5000 and testing set of size 5000. Normally distributed weight initialization with standard deviation shown in legend.

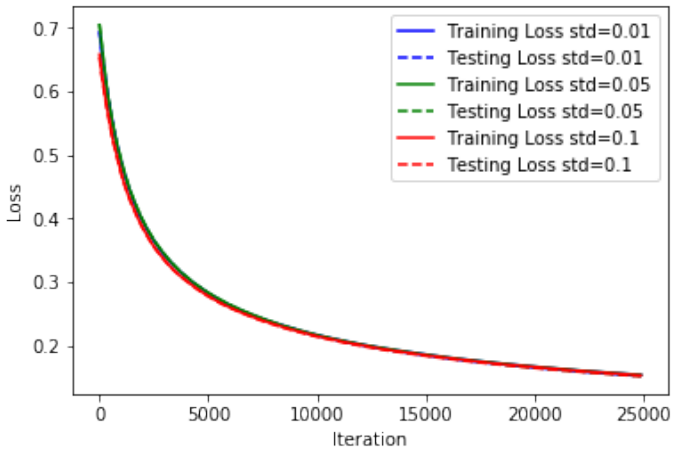



Figure 3-2: Loss over 10 epochs, learning rate = 0.001, produced by a network with no hidden layers, with 5 inputs, 2 outputs, and cross-entropy loss on linearly separable data. Used SGD with batch size of 1 and shuffled input data of size 5000 and testing set of size 5000. Normally distributed weight initialization with standard deviation shown in legend.

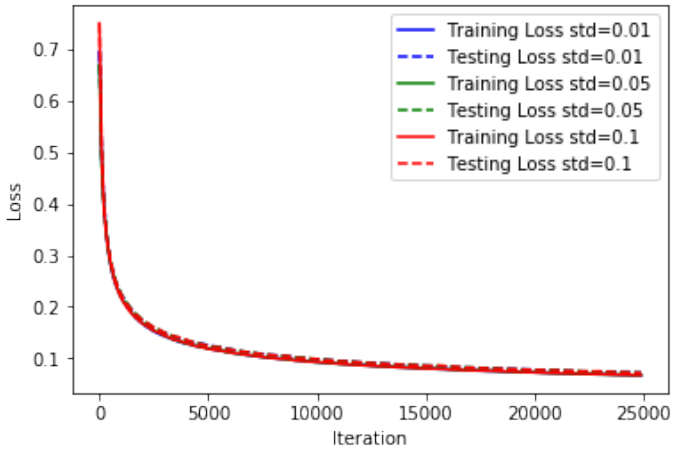



Figure 3-3: Loss over 10 epochs, learning rate = 0.01, produced by a network with no hidden layers, with 5 inputs, 2 outputs, and cross-entropy loss on linearly separable data. Used SGD with batch size of 1 and shuffled input data of size 5000 and testing set of size 5000. Normally distributed weight initialization with standard deviation shown in legend.

## 3.5 Effect of Noise perturbation on Loss

I studied the effect of adding a noise perturbation to the weights and biases in the middle of training. This effectively resets the training process each time it occurs. In Figure 2-4, the noise perturbation was introduced at epochs 3 and 30, and in Figure 2-5, the noise perturbation was introduced at epochs 25, 50, 75, 100, and 150.

## 3.6 Results with Exponential Loss Function

Then I tried these experiments again, this time using exponential loss on a network with only 1 output:

$$L(\hat{y}) = e^{-y*\hat{y}}$$

First we see in Figure 2-6 that the effect of higher standard deviation in initialization produces similar results as with cross entropy loss:

Then we look at the effect of perturbations on loss. As seen in Figure 2-7, the training and testing losses are closely related on this linearly separable dataset and after each perturbation the loss converges back. This is because adding a perturbation

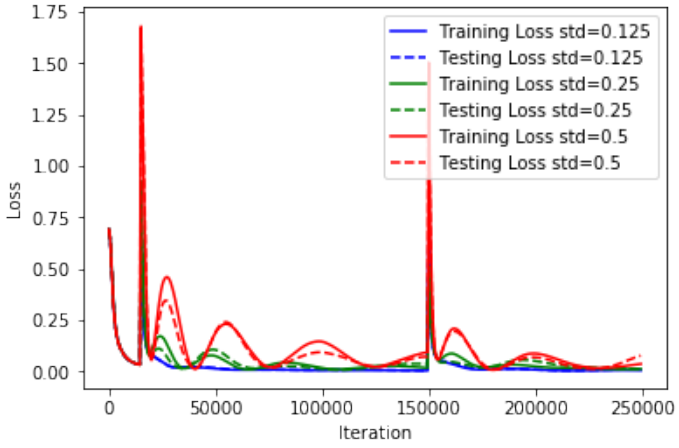



Figure 3-4: Loss over 50 epochs, with perturbations of varying standard deviation at epochs 3 and 30, produced by a network with no hidden layers, with 5 inputs, 2 outputs, and cross-entropy loss on linearly separable data. Learning rate of 0.000001 and testing and training sets of size 5000. Normally distributed weight initialization with standard deviation 0.1.

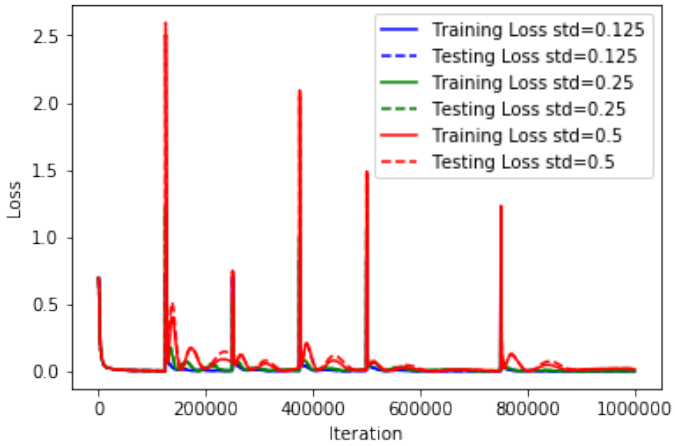



Figure 3-5: Loss over 200 epochs, with perturbations of varying standard deviation at epochs 25, 50, 75, 100, and 150, produced by a network with no hidden layers, with 5 inputs, 2 outputs, and cross-entropy loss on linearly separable data. SGD with batch size 1. Learning rate of 0.000001 and testing and training sets of size 5000. Weights initialized from normal distribution with mean 0 and standard deviation 0.1.

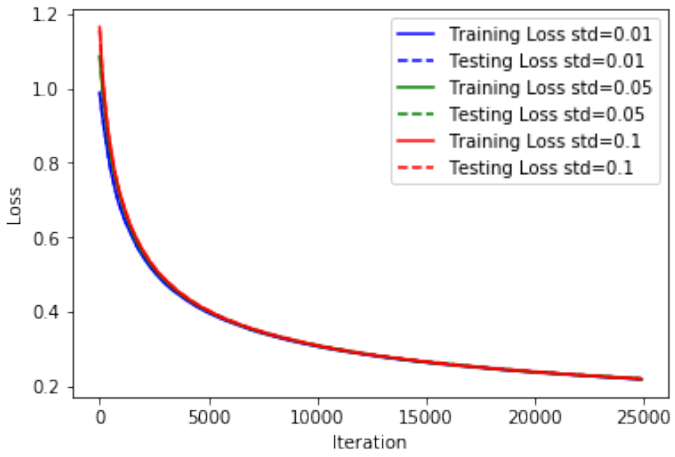



Figure 3-6: Loss over 10 epochs, learning rate = 0.0001, produced by a network with no hidden layers, with 5 inputs, 1 output, and exponential loss on linearly separable data. SGD with batch size 1. Learning rate of 0.0001 with initial standard deviation shown in legend.

is effectively resetting the training process by giving it a new starting point.

More interestingly we also see a pattern in the similarity between the original solution and the new solution after perturbation. This similarity is characterized by the cosine of the angle between the two weight vectors $\cos\theta = \frac{\mathbf{W}_0 \cdot \mathbf{W}}{||\mathbf{W}_0|| \, ||\mathbf{W}||}$. We can see that after each perturbation the new solution is very close to the original solution because the cosine is close to 1.

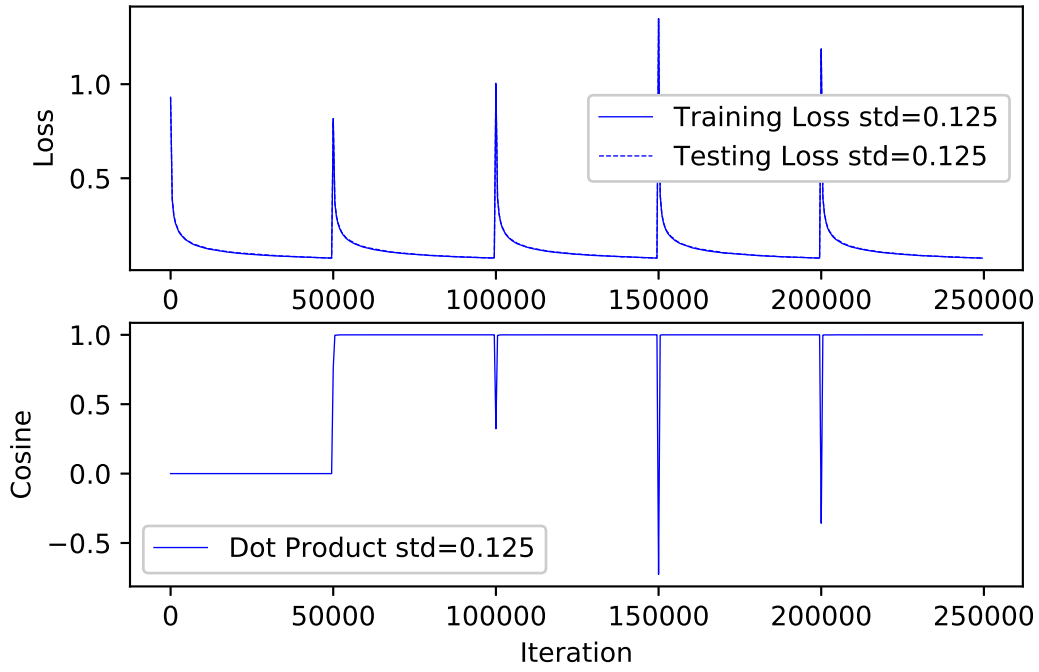



Figure 3-7: Loss over 50 epochs, with perturbations of varying standard deviation at epochs 10, 20, 30, and 40, produced by a network with no hidden layers, with 5 inputs, 1 output, and exponential loss on linearly separable data. SGD with batch size 1, learning rate of 0.01, weights initialized from normal distribution with mean 0 and standard deviation 0.1

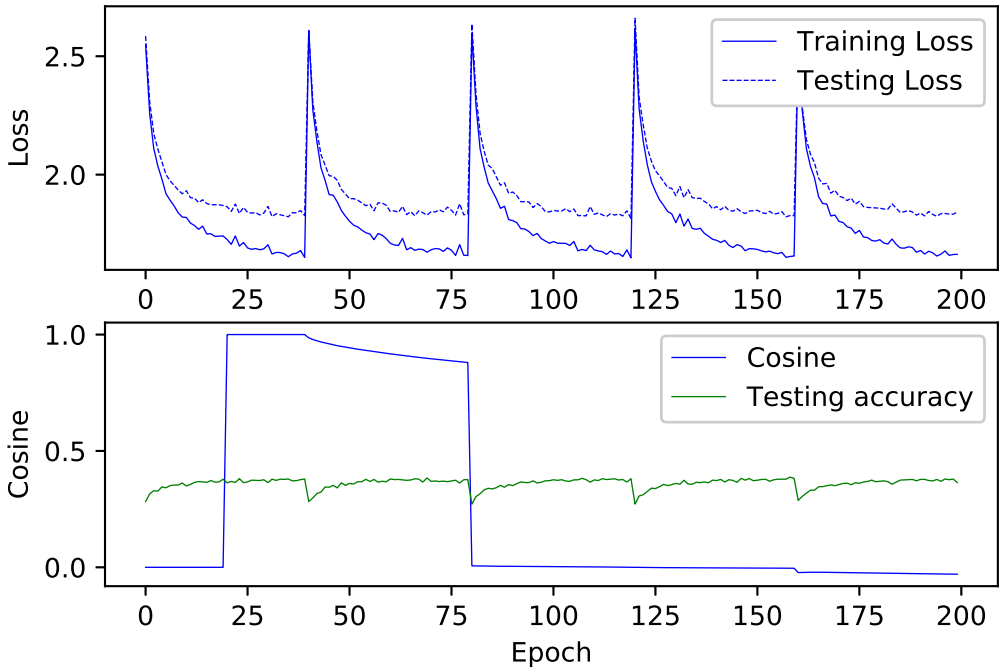



Figure 3-8: Loss, accuracy, and similarity to the first converged solution over 200 epochs, with perturbations of standard deviation 0.05 at epochs 40, 80, 120, and 160, produced by a network with no hidden layers, with 3072 inputs, 10 outputs, and cross-entropy loss on CIFAR-10. Used learning rate of 0.01, batch size of 125 and shuffled training data. The $w_0$ we compare against for the cosine was stored at epoch 20.

# Chapter 4

# Experiments Involving Weight Normalization

## 4.1 Introduction

My next task was to run a set of experiments that would produce experimental data to back up the theory specified in the paper Generalization Puzzles in Deep Networks. First, I trained ConvNets and investigated their loss and error after training, and before and after weight normalization. Then, I trained ResNets and investigated the same results, however this time normalizing the weights via a proxy method, which is dividing the weights by the norm of its output on a fixed input sample. After this, I produced more graphs for the ConvNet case involving the final product of the norms of each layer before normalization and the margin of the final classifier.

## 4.2 Confirming Theoretical Predictions for Convolutional Neural Networks

The theory only concerns overparametrized neural networks, which are neural networks with more parameters than the size of the training data. An overparametrized network can perfectly fit the training data, and a puzzle of deep neural networks

is why these overparametrized networks work well on the testing data as well. In general, statistical models and machine learning models with more parameters than training data would overfit the data and generalize poorly.

Thus, I designed a sample network with 64*5*5*3+64*5*5*64+64*8*8*10 = 148160 parameters to fit the 50K training data of CIFAR-10. This network takes in images of size 32x32 with 3 channels that goes through 3 layers, the first two convolutional, with 64 5x5 filters, stride 2, padding 2, no bias, and ReLU activation, and the third being a fully connected layer with input size 64*8*8, output size 10, no bias, and with a softmax layer before the cross entropy loss.

Some theoretical predictions I set out to confirm involved the process of weight normalization. Because there were no bias terms and I used ReLU activations, each layer could be divided by its Frobenius norm without changing the final prediction. While the final prediction (which is the label associated with the maximum output) is unchanged, this normalization will affect the cross entropy loss. The theory shows that what is expected is that the testing loss and training loss after normalization will be linearly related with an offset. As shown in the later images, I found that this was true.

## 4.3 Investigating Theoretical Predictions for Deep Residual Networks

My next task was to find a working ResNet implementation and train it on CIFAR-10. I tested a few GitHub repositories with ResNet implementations before choosing junyuseu's Pytorch implementation due to its great performance and training speed.

During this process of looking for a good implementation, I confirmed that the strategy of using decreasing learning rates during SGD was effective for reaching a final network with good testing error. When training with 0.1 learning rate the testing error gets stuck around 0.11 while when training further with 0.001 and 0.0001 learning rate brings the testing error down to 0.08.

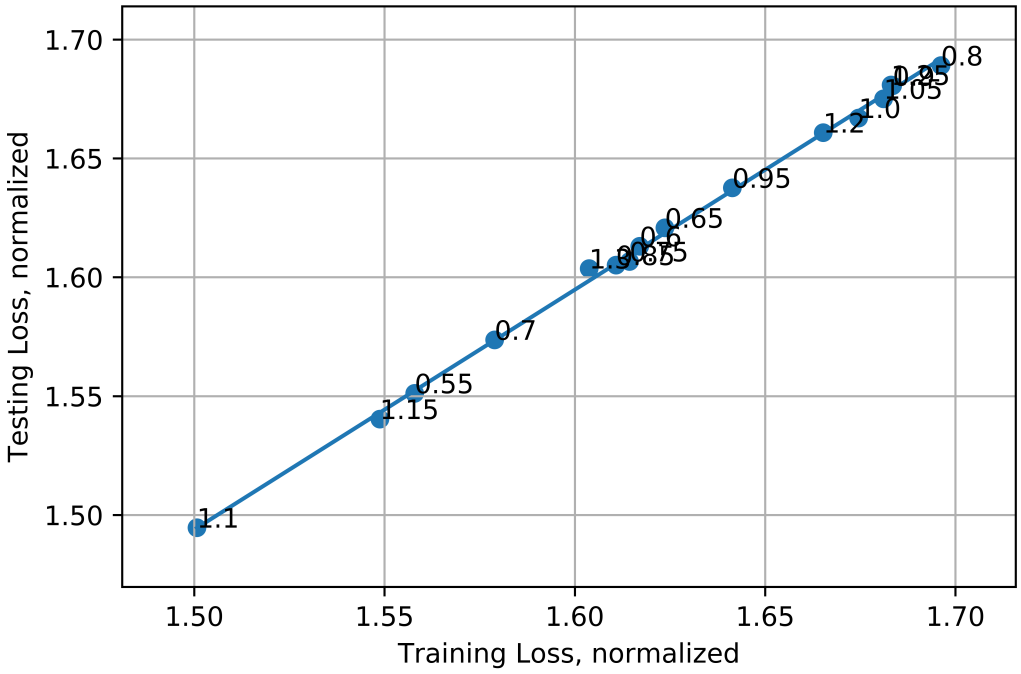



Figure 4-1: *Final training loss vs. testing loss for networks initialized with weight scaling according to label. Trained for 200 epochs with learning rate = 0.01 for the first 100 epochs and learning rate = 0.001 for the second 100 epochs, produced by a 56-layer ResNet as detailed in Deep Residual Learning for Image Recognition by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, with cross-entropy loss on CIFAR-10 . Used SGD with batch size of 128 and shuffled training data with random crop data augmentation.*

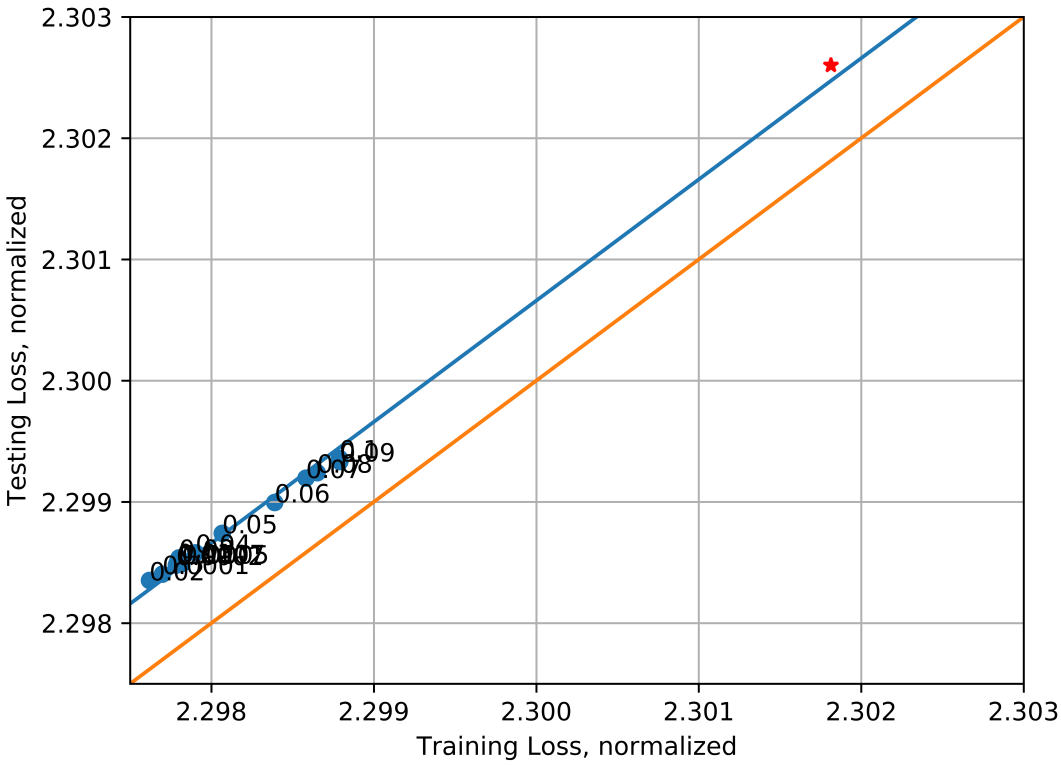



Figure 4-2: *3-layer: Training Loss vs. Testing Loss after normalizing each layer by its Frobenius norm, after training to 0 training error. 3-layer convnet (two convolutional, each with 32 5x5 filters, stride 2, padding 2, no biases, and ReLU activation, and one linear layer with input 32x8x8, 10 outputs, no biases, softmax activation). No data augmentation. Trained for 200 epochs with decaying learning rate (0.01, 0.01, 0.001, 0.0001). Used SGD with batch size 128.*

Also, to reach that low of a testing error, data augmentation is crucial. Data augmentation in PyTorch works by randomly cropping and horizontally flipping the training data at each iteration over the data, which means the training set is increased by a huge amount. These transformations should in practice not damage the images sufficiently to make them unrecognizable. Thus the training set is greatly augmented with new data which increases the performance.

The following graphs were included in a Nature submission and show that the training loss can be a good proxy for the test loss if we compare using normalized weights at each layer.

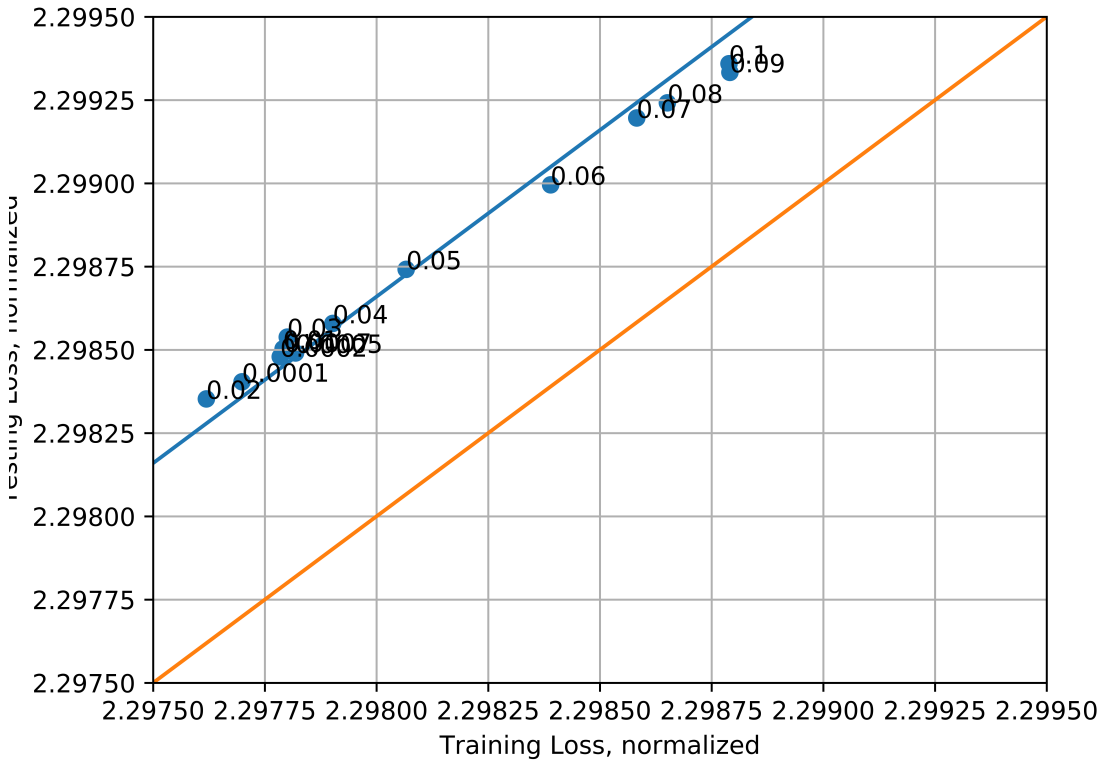



Figure 4-3: *3-layer: Zoomed-in training Loss vs. Testing Loss after normalizing each layer by its Frobenius norm, after training to 0 training error. 3-layer convnet (two convolutional, each with 32 5x5 filters, stride 2, padding 2, no biases, and ReLU activation, and one linear layer with input 32x8x8, 10 outputs, no biases, softmax activation). No data augmentation. Trained for 200 epochs with decaying learning rate (0.01, 0.01, 0.001, 0.0001). Used SGD with batch size 128.*

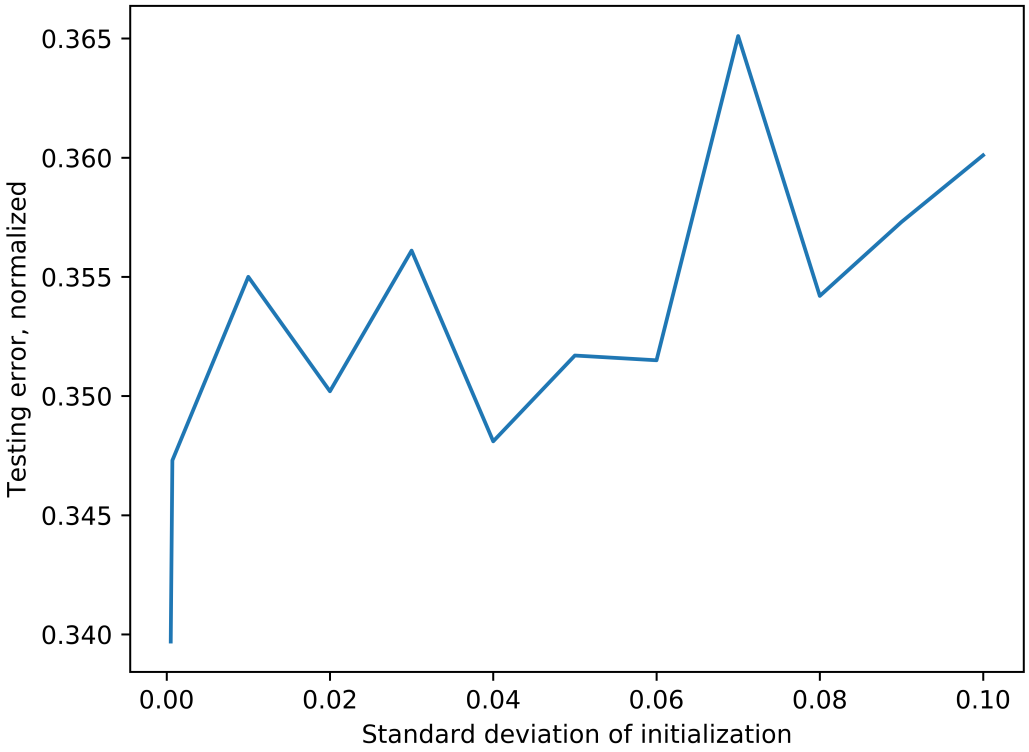



Figure 4-4: *Standard deviation of initialization vs. final testing error after training network to 0 training error. 3-layer convnet (two convolutional, each with 32 5x5 filters, stride 2, padding 2, no biases, and ReLU activation, and one linear layer with input 32x8x8, 10 outputs, no biases, softmax activation). No data augmentation. Trained for 200 epochs with decaying learning rate (0.01, 0.01, 0.001, 0.0001). Used SGD with batch size 128.*

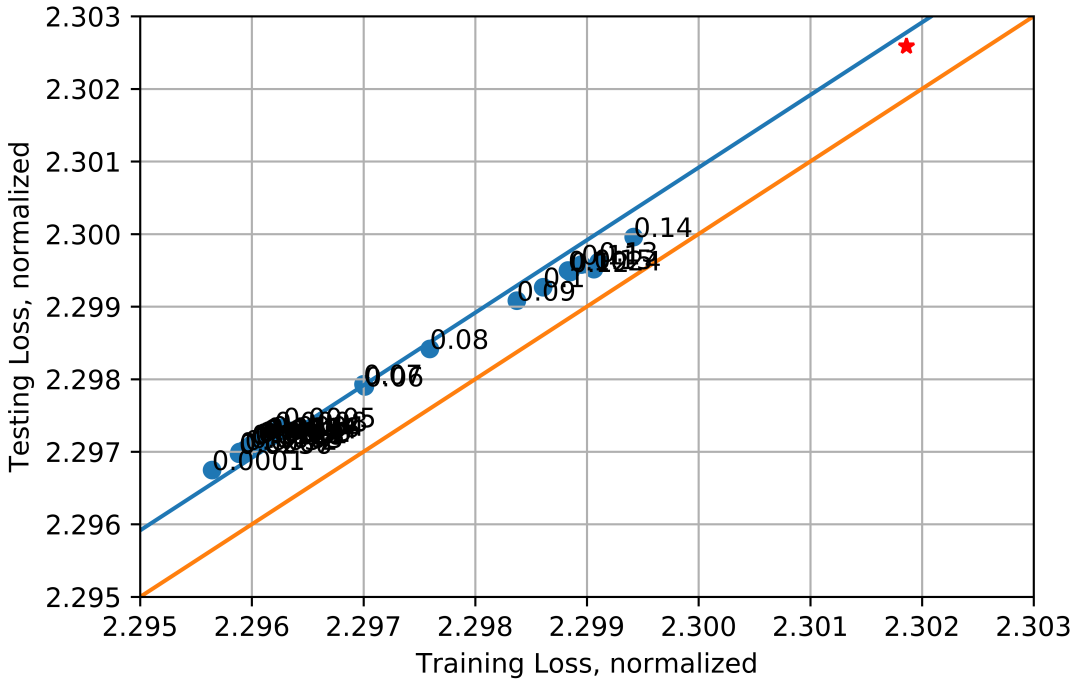



Figure 4-5: *3-layer: Standard deviation of initialization vs. final testing error after normalizing each layer by its Frobenius norm. First layer: Convolutional, 64 filters of size 5x5, stride 2, padding 2, no bias, ReLU activation. Second layer: Convolutional, 64 filters of size 5x5, stride 2, padding 2, no bias, ReLU activation. Third layer: Fully connected, input size 64*8*8, output size 10, no bias, linear activation. Used cross-entropy loss on CIFAR-10 dataset: 50k training examples, 10k testing examples. No data augmentation, but data were normalized to mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2023, 0.1994, 0.2010). Trained for 200 epochs with decreasing learning rate (0.01, 0.01, 0.001, 0.0001) until training error reached 0 percent. The weights were initialized from a normal distribution with mean 0 and standard deviation in [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001, 0.0016, 0.0032, 0.0064, 0.0128, 0.0256, 0.0512, 0.06, 0.07, 0.08, 0.09, 0.1, 0.1024, 0.11, 0.12, 0.13, 0.14, 0.15]. RL point was trained similarly except the testing set and training set labels were randomized. For the RL point, the weights were initialized from a normal distribution with mean 0 and standard deviation 0.1.*

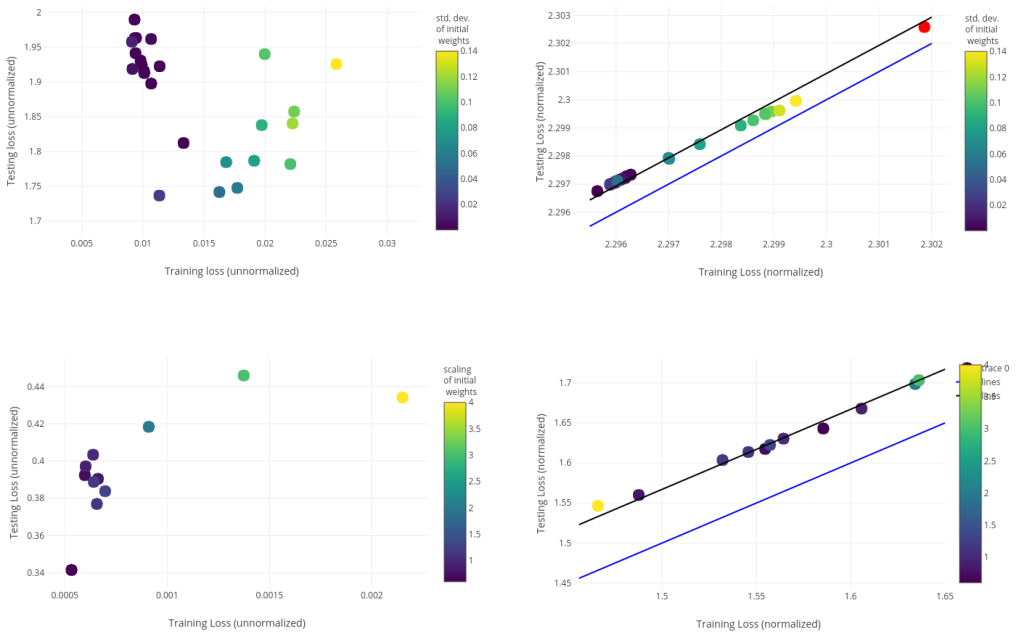
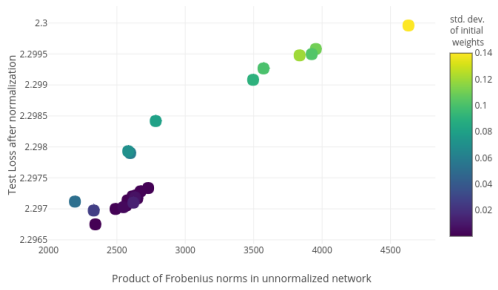
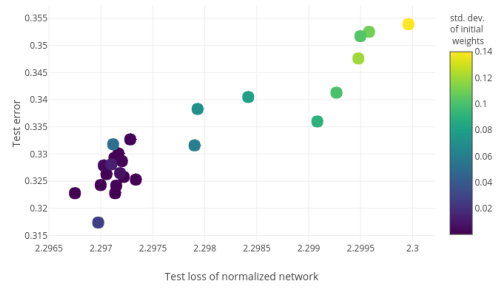


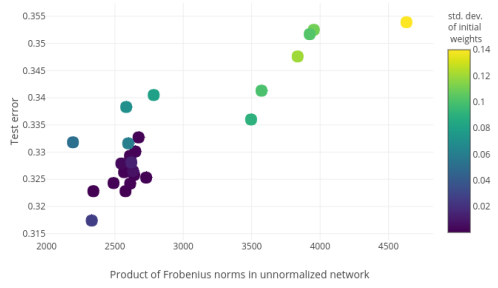

Figure 4-6: *Top left graph shows testing vs training cross-entropy loss for networks trained on the same data sets but with different initializations. The top right graph shows the testing vs training loss for the same networks, normalized by dividing each weight by the Frobenius norm of its layer. The RL point refers to a network trained on the same CIFAR data set but with randomized labels. It shows zero classification error at training and test error at chance level. The line shows the regression of slope* 1 *with a positive intercept. The networks are 3-layer; the first two layers are convolutional, with 64 filters of size 5x5, stride 2, padding 2, no bias, ReLU activation; the third layer is fully connected, input size 64*8*8, output size 10, no bias, softmax activation. Trained on the whole CIFAR-10 dataset. No data augmentation, but data were normalized to mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2023, 0.1994, 0.2010). Trained for 200 epochs with decreasing learning rate (0.01, 0.01, 0.001, 0.0001) until zero training error. The weights were initialized from a normal distribution with mean 0 and standard deviation in [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001, 0.0016, 0.0032, 0.0064, 0.0128, 0.0256, 0.0512, 0.06, 0.07, 0.08, 0.09, 0.1, 0.1024, 0.11, 0.12, 0.14]. The network for the RL point was trained similarly except the testing set and training set labels were randomized. For the RL point, the weights were initialized from a normal distribution with mean 0 and standard deviation 0.1. The bottom graphs show similar data for a ResNet. The normalization in this case was performed by using the norm of the output of each network on one example from CIFAR-10. The networks were trained for 200 epochs with learning rate = 0.01 for the first 100 epochs and learning rate = 0.001 for the second 100 epochs, produced by a 56-layer ResNet as detailed in Deep Residual Learning for Image Recognition by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. SGD was used with batch size of 128 and shuffled training data with random crop and horizontal flip data augmentation.*




(a)




(b)




(c)

Figure 4-7: *a) Test loss for the normalized networks vs the product of the unnormalized Frobenius norms of the layers, b) test classification error as a function of the normalized test loss and c) test classification error vs the product of the unnormalized Frobenius norms of the layers. The networks are 3-layer; the first layer is convolutional, 64 lters of size 5x5, stride 2, padding 2, no bias, ReLU activation; the second layer is also convolutional, 64 lters of size 5x5, stride 2, padding 2, no bias, ReLU activation; the third layer is fully connected, input size 64*8*8, output size 10, no bias, linear activation. Trained on CIFAR-10 dataset: 50k training examples, 10k testing examples. No data augmentation, but data were normalized to mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2023, 0.1994, 0.2010). Trained for 200 epochs with decreasing learning rate (0.01, 0.01, 0.001, 0.0001) until training error reached 0 percent. The weights were initialized from a normal distribution with mean 0 and standard deviation in [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001, 0.0016, 0.0032, 0.0064, 0.0128, 0.0256, 0.0512, 0.06, 0.07, 0.08, 0.09, 0.1, 0.1024, 0.11, 0.12, 0.14].*

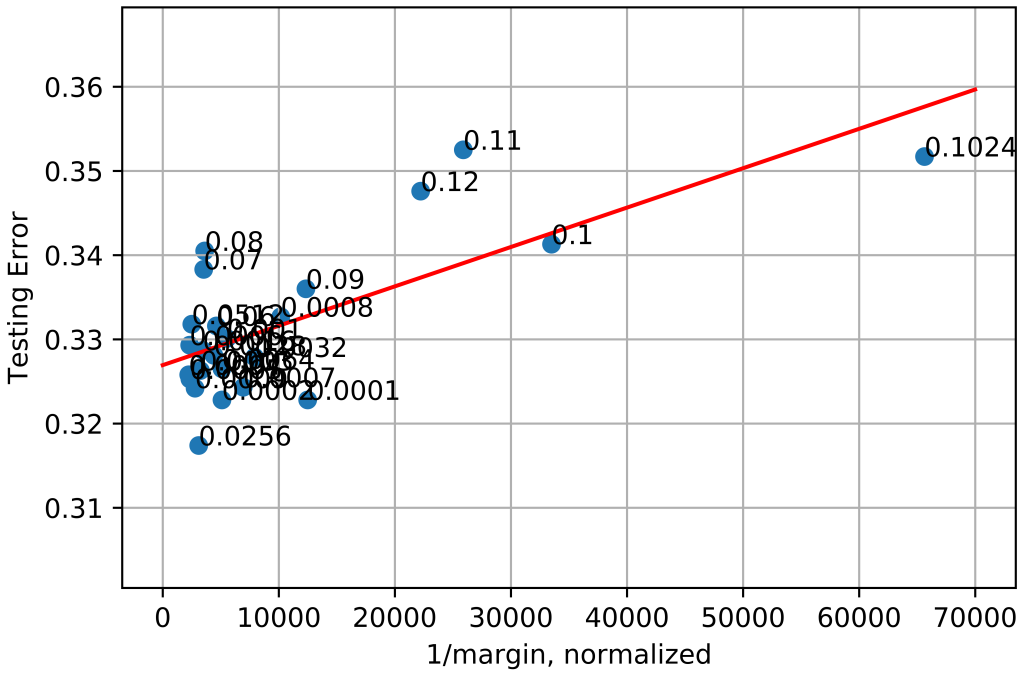



Figure 4-8: *Final testing error versus 1/margin of the normalized network. The network is 3-layer; the first layer is convolutional, 64 lters of size 5x5, stride 2, padding 2, no bias, ReLU activation; the second layer is also convolutional, 64 lters of size 5x5, stride 2, padding 2, no bias, ReLU activation; the third layer is fully connected, input size 64*8*8, output size 10, no bias, linear activation. Trained on CIFAR-10 dataset: 50k training examples, 10k testing examples. No data augmentation, but data were normalized to mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2023, 0.1994, 0.2010). Trained for 200 epochs with decreasing learning rate (0.01, 0.01, 0.001, 0.0001) until training error reached 0 percent. The weights were initialized from a normal distribution with mean 0 and standard deviation in [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001, 0.0016, 0.0032, 0.0064, 0.0128, 0.0256, 0.0512, 0.06, 0.07, 0.08, 0.09, 0.1, 0.1024, 0.11, 0.12, 0.14].*

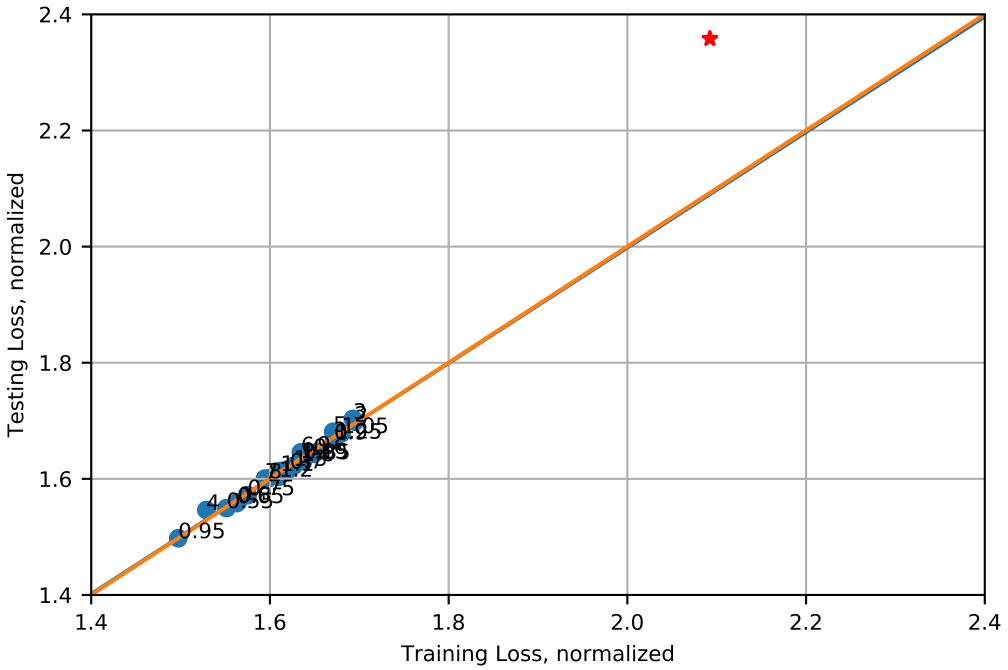



Figure 4-9: *Final testing error versus 1/margin of the normalized network. The network is a 56-layer ResNet as detailed in Deep Residual Learning for Image Recognition by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, with cross-entropy loss on CIFAR-10 . Used SGD with batch size of 128 and shuffled training data. Training data were augmented with random cropping and horizontal flipping at every epoch. Immediately after the initialization the weights were scaled by the scaling factors shown next to the points on the graph. The network for the RL point was trained similarly except the testing set and training set labels were randomized. For the RL point, the weights were initialized from a normal distribution with mean 0 and standard deviation 0.1.*

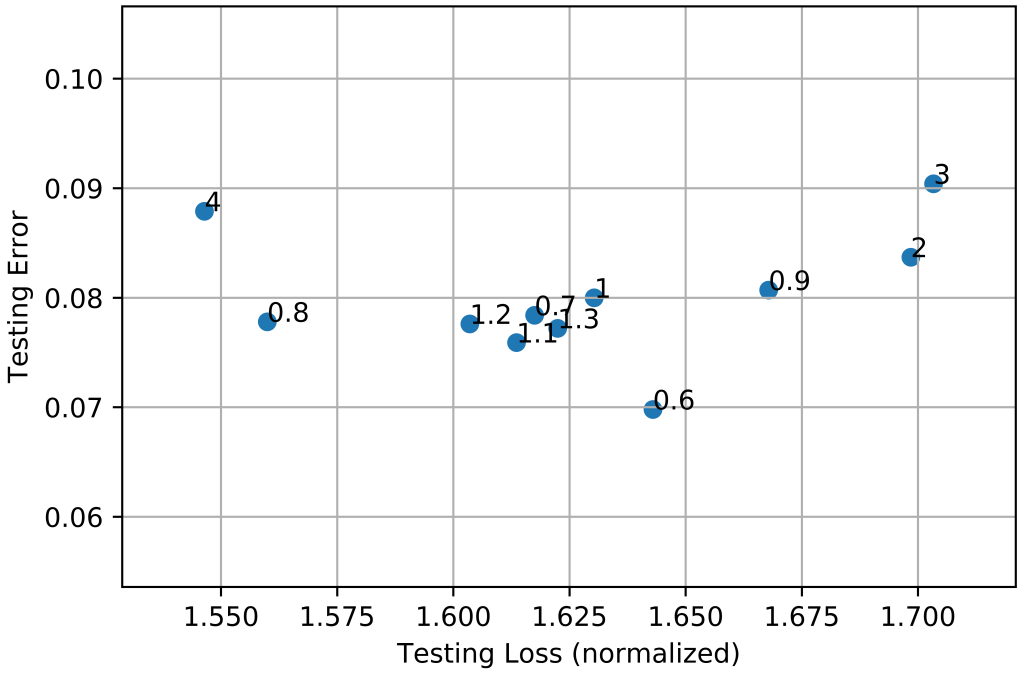



Figure 4-10: *Final testing error versus testing loss of the normalized network. The network is a 56-layer ResNet as detailed in Deep Residual Learning for Image Recognition by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, with cross-entropy loss on CIFAR-10 . Used SGD with batch size of 128 and shuffled training data. Training data were augmented with random cropping and horizontal flipping at every epoch. Immediately after the initialization the weights were scaled by the scaling factors shown next to the points on the graph.*

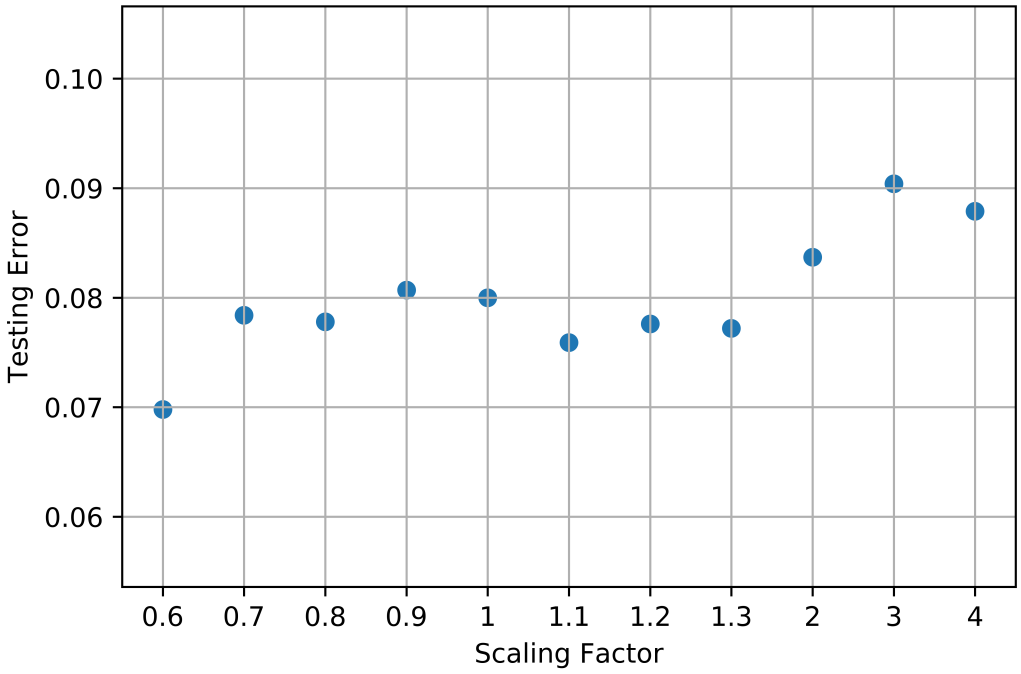



Figure 4-11: *Final testing error versus scaling factor used of the normalized network. The network is a 56-layer ResNet as detailed in Deep Residual Learning for Image Recognition by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, with cross-entropy loss on CIFAR-10 . Used SGD with batch size of 128 and shuffled training data. Training data were augmented with random cropping and horizontal flipping at every epoch. Immediately after the initialization the weights were scaled by the scaling factors.*

# Chapter 5

# Further Work

## 5.1 Summary

Now that we can see that the normalized weights of a network are the true parameters we care about, we look towards methods of training that reflect this observation. First, I tried manually normalizing weights to 1 after each epoch. Also, I tried training with absorbing these norms into rho, which in the normal case is the product of norms of each layer.

After that, I started work on using a Lagrange multiplier method, where we train using a different loss function

$$L = \sum_{i=1}^{N} L_i + \sum \lambda_k^2 (||V_k|| - 1)^2$$

where the $L_i$ are calculated with the desired loss function, in our case the cross entropy loss on CIFAR-10 labels, while we impose a new constraint to keep the norm of $V_k$ to be 1. This work is in its starting stages but could lead to a second paper.

# Bibliography


[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *arXiv e-prints*, page arXiv:1512.03385, 2015.

[2] T. Poggio and Q. Liao. Theory II: Landscape of the Empirical Risk in Deep Learning. *arXiv:1703.09833, CBMM Memo No. 066*, 2017.

[3] T. Poggio, Q. Liao, B. Miranda, L. Rosasco, X. Boix, J. Hidary, and H. Mhaskar. Theory of Deep Learning III: Explaining the non-overfitting puzzle. *arXiv:1703.09833, CBMM Memo No. 073*, 2017.

[4] T. Poggio, Q. Liao, B. Miranda, A. Banburski, L. Rosasco, R. Liang, and J. Hidary. Theory IIIb: Dynamics and Generalization in Deep Networks. *arXiv:1703.09833, CBMM Memo No. 090*, 2019.

[5] B. Neyshabur, S. Bhojanapalli, D. McAllester, N. Srebro. Exploring Generalization in Deep Learning. *arXiv:1706.08947*, 2017.

[6] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer Automatic differentiation in PyTorch. *Neural Information Processing Systems*, 2017.

[7] Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering Volume 9*, 2007.