Multi-Layer Neural Networks

Christian Koguchi
PID: A10910223

Daniel Reznikov
PID: A12440386

Abstract

In this report, we explore the "tricks of the Neural Network trade" starting from our softmax regression solution to the previous homework. We extend our network to support different activation functions, weight initializations, and network topologies. We achieve nearly state-of-the art results with our best models hitting 99.7% accuracy on the validation set and 97.4% on the independently sourced test set.

Introduction

The dataset that we are trying to classify is the classic MNIST dataset (sourced from Yann LeCun)[**?**]. We aim to explore network topologies of various configurations for multi-layer perceptrons. We implement efficient backpropogation of gradient descent manually. We explore many common tricks of the trade and their sensitivity to hyperparameters and initial conditions. In sum, this work crystallized our understanding of foundational neural network concepts. What follows is an instructive discussion about our learning.

Methodology

Classification

- ***Lazy-Man Normalization:*** Pre-process the images by dividing by 127 and subtract 1, thereby moving all pixel values to the ranges [-1, 1].
- ***Hidden Layer***: initialize the network with 1 hidden layer with 64 hidden units.
- ***Numerical Gradient Checking***: to validate our backpropogation algorithm implementation, we compare individual weight updates against the numerical approximation.
- ***Mini-Batch***: we expose the hyperparameter *batch-size* that allows us to separate the training set into small batches. We update and calculate the gradient per batch. We explore the impact of mini-batching by analyzing our accuracy scores on a withheld validation set of 10,000 randomly selected training images.
- ***Grid Search***: we implement brute-force grid search by manually searching over the space of learning rates and annealing parameters. We do a search over the log-space characterized by the following intervals: $\{\eta \in [0.01, 0.00001],\ \ T \in [1, 1000]\}$.

Tricks of the Trade

- ***Performance Metrics:*** We select a 97% accuracy as a good threshold to benchmark our models. We chose our two statistics because they independently measure interesting behavior. Wall-Clock time is intrinsically valuable to the end-user because it represents the real cost to training. Epochs to train give insight into how the experiment setup is able to lend itself to the model converging.
- ***Shuffling After Epochs***: we randomly permute our training data before splitting into mini-batches.
- ***Tanh Hidden Activation Function***: to compare different members of the sigmoid class of functions, we implement the hyperbolic tangent function and use it for the activation for our hidden nodes. These experiments were highly sensitive to initial weight conditions, so we invested great effort in tuning initial conditions. The motivation for the tanh function is so that the hidden layer outputs have an average value of approximately 0 whereas the sigmoid will not.
- ***Initial Weight Setup***: we further our analysis by initializing our weights to a normal distribution with mean = 0 and standard deviation = $\frac{1}{\sqrt{\text{fan-in}}}$. The fan-in is the number of connections from the previous layer.
- ***Momentum***: leveraging the intuition from Lecun's '98 paper on momentum, we implement a momentum term that increases the step of the gradient with a term that is proportional to the running average of the gradients in the previous iterations.

Experiments with Network Topology

- ***Hidden Unit Exploration***: we halve the number of hidden units to 32 and double the hidden units to 128. We profile our results by again measuring the wall-clock time and epochs to achieve a validation accuracy of 97%.
- ***Hidden Layer Exploration***: We add another hidden layer with the same number of hidden nodes (64) as the first layer. Both layers have a bias term that is not connected to the previous layer.

Results

Numerical Gradient Checking

To validate backpropagation, we imploy a hook in our code that computes the numerical gradient as follows:

$$\frac{\partial E^n}{\partial w_{ij}} \approx \frac{E^n(w_{ij} + \epsilon) - E^n(w_{ij} - \epsilon)}{2\epsilon}$$

We test with $\epsilon = 0.01$. We found that the difference between the numerically computed gradients and the backprop gradients are between $[10^{-5},\ 10^{-7}]$ for various model parameters and we found that the difference is always $O(\epsilon^2)$, as expected.

Table 1: Numerical Gradient Checking

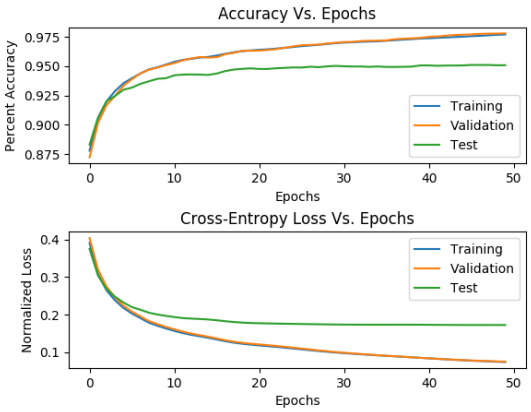
$\epsilon = 10^{-2}$	Numerical Gradient	Backprop Gradient	Difference	$O(\epsilon^2)$
Input Bias(0, 63)	-0.027	-0.028	-1.855$\times 10^{-7}$	T
Input Bias(0, 4)	0.311	0.311	-6.823$\times 10^{-6}$	T
Output Bias(0, 3)	-10.082	-10.083	8.523$\times 10^{-5}$	T
Output Bias(0, 4)	-1.310	-1.311	9.444$\times 10^{-5}$	T
w_{hidden}(3, 4)	0.322	0.322	3.685$\times 10^{-7}$	T
w_{hidden}(64, 63)	-0.310	-0.309	5.960$\times 10^{-7}$	T
w_{output}(2, 2)	-0.364	-0.364	4.226$\times 10^{-6}$	T
w_{output}(37, 7)	1.138	1.382	5.120$\times 10^{-5}$	T

Experiment Results

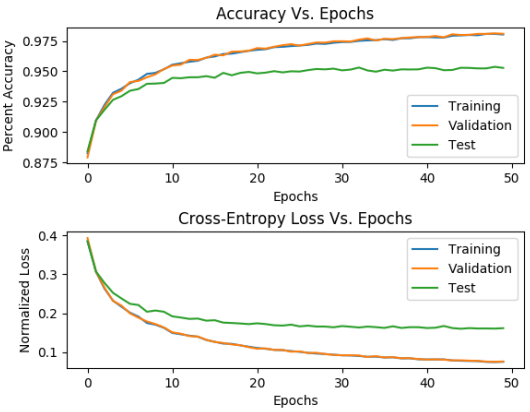
We found that all configurations of our experiments can be tuned to achieve very high accuracies. Our results follow:

Table 2: Results to get 97% Validation Accuracy

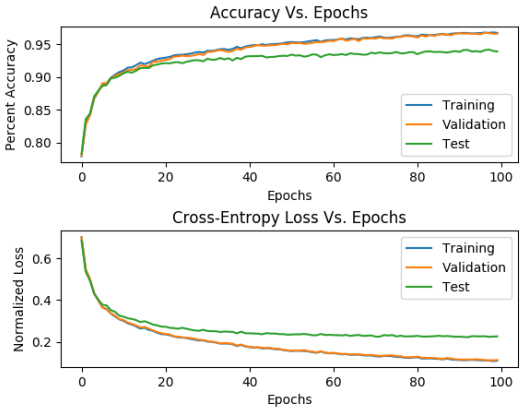
Experiment No.	**Description**	**Wall-Clock Time [secs]**	**Epochs**
1	Mini-Batch of 128 + Weights from a Uniform Distribution[-1, 1]	31.74	30
2	Shuffling after Epochs and Averaging the Gradient	29.57	22
3	Using Tanh Activation Function	153.70	99
4	Initialize Weights $w \sim \mathcal{N}\left(0,\ \frac{1}{\sqrt{fan-in}}\right)$	10.60	6
5	Momentum α = 0.9	10.95	6
6	Halving Hidden Units to 32	14.27	12
7	Doubling Hidden Units to 128	17.16	6
8	Using 2 Hidden Layers	9.03	24



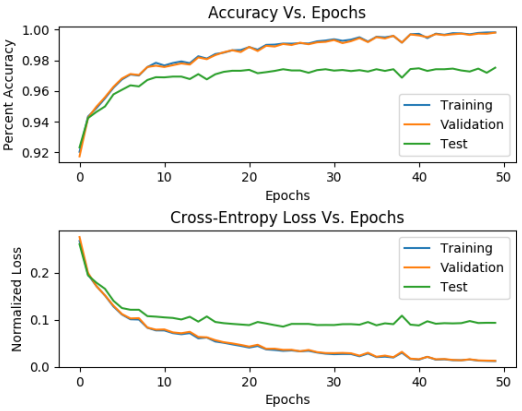
Experiment 1: Mini-batching 128



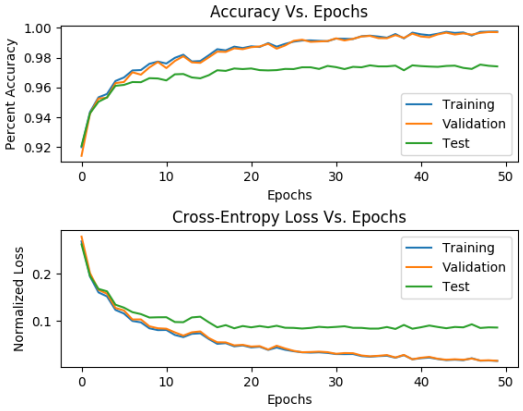
Experiment 2: Shuffling after Epochs/Averaging



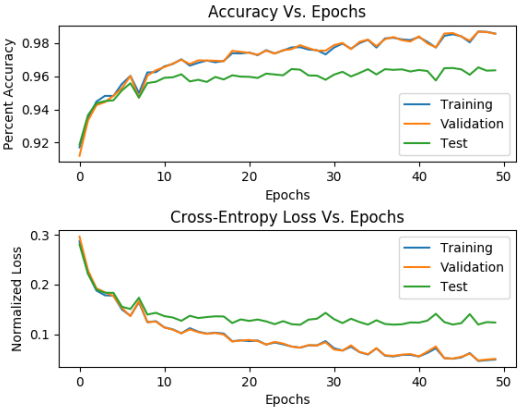
Experiment 3: Tanh Activation Function



Experiment 4: Initializing Weights using Fan-In



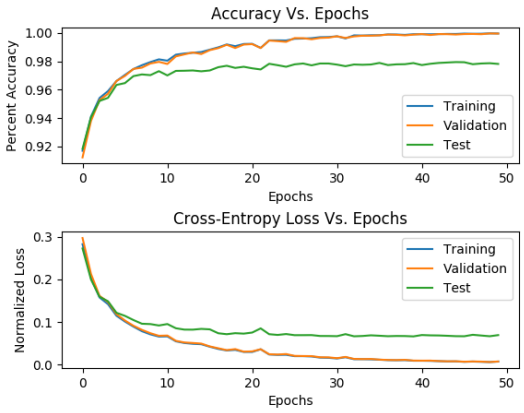
Experiment 5: Momentum



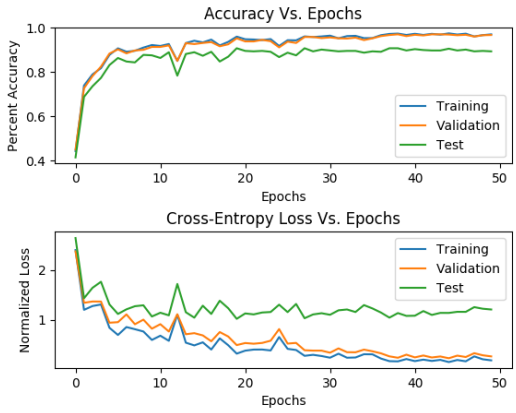
Experiment 6: Halving Hidden Units

Discussion

- ***Mini-Batch***: We are pleasantly surprised by how mini-batching greatly accelerates the rate of learning. By exploiting the stochastic numerical processes, we are able to converge rapidly upon an optimal solution due to the redundancy in the training set. This made



Experiment 7: Doubling Hidden Units



Experiment 8: Two Layer Network

training our large dataset more manageable. While we didn't experiment much with the batch size, we found that the initial setup of 128 was already a good balance between performance and being large enough to provide a meaningful sample of our training data.

- ***Shuffling***: To further exploit the randomness in our stochastic approach, we introduce shuffling after each epoch. This means our learning is more robust because the class distribution is rigorously randomized within mini-batches. Our results show that while the wall-clock time was stable, the number of epochs decreased by 50%.
- ***Tanh***: Yann LeCun's paper developed our intuition as to why the tanh function will be more efficient during learning. In comparison to the logistic sigmoid, the output weights of the tanh activation function will on average gravitate towards 0 whereas the range of the logistic sigmoid is always positive. During backpropagation, this means that the weight change rules can move in both positive and negative directions. We found that the tanh function was markedly better, achieving the 97% accuracy in 6 epochs as compared to 22 and in 10.60 seconds.
- ***Momentum***: The loss function is locally convex, so the intution behind the momentum term is that it will push our model over local minima allowing it to converge faster and with higher accuracy. Despite this intuition, we did not find significant improvements in either of our performance metrics (see table above). We suspect this is happening because our other hyperparameters are already compensating sufficiently for fast convergence.
- ***Halving Hidden Units***: Halving the number of hidden units does not decrease the model accuracy but increases the wall-clock time by about 50% and the number of epochs by about 100%. From this we conclude that it was not by happenstance that we were asked to use 128 hidden units because that seems to be the sweet spot.
- ***Doubling Hidden Units***: Doubling the number of units in the hidden layer to 128 resulted in the same training epochs (8) but more wall-clock time (50% increase). Doubling the number of hidden units doubles the number of weights. The number of training epochs doesn't change because multiple neurons may be learning the same features.
- ***Using 2 Hidden Layers***: Our intuition told us that two hidden layers should train longer but produce better models. Our hypothesis was confirmed by obversving that we can achieve near state-of-the-art performance while epochs remained the same and training time in seconds doubles.

Conclusion

First, we are very grateful for this assignment because we have learned many interesting intuitions about neural networks. The most profound learnings were the influence of the tanh activation function and the sensitivity of these networks to weight initializations. We appreciated deeply that each

subsequent experiment clearly demonstrated improvements over each predecessor. Lecun's whitepaper and our lectures greatly enhanced our intuition about network topologies and tricks of the trade. The rigorous mathematical analysis requried to both apply the recursive backpropagation was a rewarding challenge. In sum, we are very pleased to report models that achieve 97.4% validation set accuracy.

Individual Contributions

Christian did much of the heavy lifting for the derivation for backprop. Christian was instrumental in vectorizing our code and to prototyping backpropagation. Christian helped with the write-up. Christian implemented grid-search to find the optimal parameters for our models.

Daniel architected our code to be modular, reusable, and professional. He enforced the Extreme-Programming paradigm and code style. Daniel spent much effort on debugging and training our models. Daniel worked extensively on the write-up.

Appendix

For reproducibility, we provide our model hyperparameters as follows.

Table 3: Model Hyperparameter Values

Experiment No.	**Annealing**	**Learning Rate**	**Shuffle**	**Hidden Activation**	**Weight Init.**	**Momentum**	**Hidden Nodes**	**Hidden Layers**
1	100	0.01	False	Logistic	uniform [-0.5, 0.5]	False	64	1
2	10	0.01	True	Logistic	uniform [-0.5, 0.5]	False	64	1
3	1	0.001	True	Tanh	uniform [-0.5, 0.5]	False	64	1
4	600	0.001	True	Tanh	$\mathcal{N}\left(0, \frac{1}{\sqrt{\text{fan-in}}}\right)$	False	64	1
5	10	0.0005	True	Tanh	$\mathcal{N}\left(0, \frac{1}{\sqrt{\text{fan-in}}}\right)$	True	64	1
6	100	0.00005	True	Tanh	$\mathcal{N}\left(0, \frac{1}{\sqrt{\text{fan-in}}}\right)$	True	32	1
7	750	0.00005	True	Tanh	$\mathcal{N}\left(0, \frac{1}{\sqrt{\text{fan-in}}}\right)$	True	128	1
8	1	1	True	Tanh	$\mathcal{N}\left(0, \frac{1}{\sqrt{\text{fan-in}}}\right)$	True	64	2