A Deep Dive on the Matrices of a Neural Network

Darren Yu
University of California, San Diego
La Jolla, CA 92093
dmyu@ucsd.edu

January 25, 2024

Abstract

The fundamental principles of neural networks were applied to a small-scale model to classify digits in a base$_{10}$ numeral system. An array of hyperparameters, e.g., learning rate, momentum, and regularization penalties, were carefully picked to fine-tune a neural network model. From my multi-layer model, I found that using the sigmoid activation in the hidden layer with a small learning step ($\alpha = 0.0001$) yielded the best validation set accuracy while minimizing the cross-entropy loss. The least absolute shrinkage and selection operator (Lasso Regression) was used to regularize my loss to avoid overfitting to my training set. These key optimizations allowed my model to correctly identify 1,000 images of 10,000 elements in the test set. Thus, my final model yielded 97.88% accuracy.

1 Exploratory Analysis

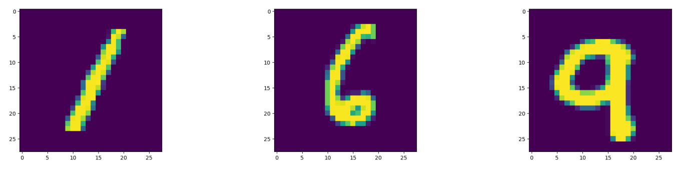
I chose to work on the MNIST dataset, which contained 70,000 images of individual base$_{10}$ digits. The ground truth was an integer representing the digit of each corresponding image. To begin, I split my data into a 48,000 train set, a 12,000 validation set, and a 10,000 test set.

1.1 Data Normalization

From each image, I computed the mean and standard deviation and applied a z-score to each pixel. Normalizing each pixel between -1 and 1 allows a neural network to learn faster and minimize out-of-scale values. The ground truth labels of each corresponding image were converted into a one-hot encoded form such that the index of the hot bit signifies the specified label.

1.2 Image Depictions

After normalizing the data, I retrieved the first three elements from the train set and plotted them as some data points. Here are the plots with their corresponding encoding and values:



(a) 0 1 0 0 0 0 0 0 0 0 (b) 0 0 0 0 0 0 1 0 0 0 (c) 0 0 0 0 0 0 0 0 0 1

Figure 1: Three Sample Data Points

Image:	Mean:	Standard Deviation:
A	0.07959	0.24883
B	0.11058	0.28719
C	0.19310	0.36961

Table 1: Mean and Standard Deviation

2 Models

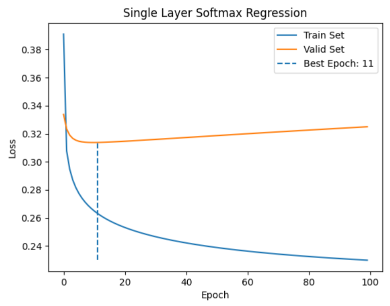
A collection of neural network models were created and yielded varying results. Test accuracy ranged from 90% to 98% prediction accuracy. The choice of layer depth, learning rate, and activation functions were modified to generate multiple high-performing models.

2.1 Baseline

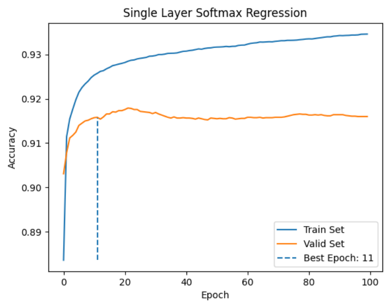
To start with a reference model, I constructed a 784 input to 10 output neural network. The output layer would be a probability distribution of each class generated from softmax regression. With only a single layer, the update rule for this particular model was $w_{ij} = w_{ij} + (t_j - y_j)z_i$. The following are the hyperparameters and the model results:

α	b_{size}	ϵ	ϵ_{stop}	Test Accuracy	Stoppage
0.001	128	100	3	0.9181	11

Table 2: Single Layer Parameters and Results



(a) Cross Entropy Loss



(b) Accuracy

Figure 2: Single Layer Softmax Regression

2.2 Backpropagation

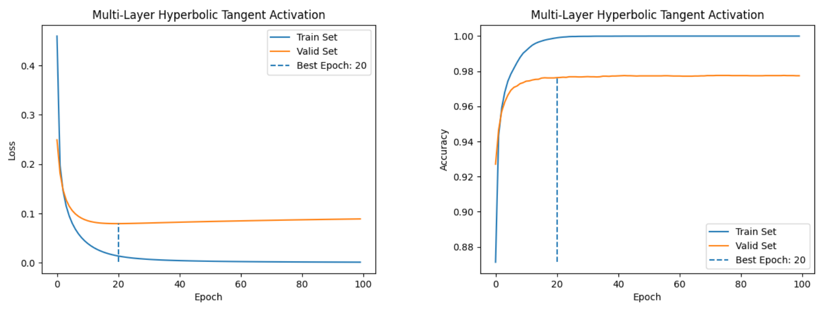
I extended my baseline model with an additional hidden layer containing 128 neurons. The new model now had 784 input to 128 hidden to 10 output. I tried different activation functions in the hidden layer, like $tanh$, $sigmoid$, and $ReLU$. With the addition of a hidden layer, there were now two update rules: $w_{ij} = w_{ij} + (g'(a_j)\sum_k \delta_k w_{jk})z_i$ and $w_{jk} = w_{jk} + (t_k - y_k)z_j$.

2.2.1 Hyperbolic Tangent Activation

The hyperbolic tangent activation function ($tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$) was an excellent choice because of its ability to map values between [-1, 1]. The following are the hyperparameters of the multilayer model:

α	$g(a)$	b_{size}	ϵ	ϵ_{stop}	Test Accuracy	Stoppage
0.001	tanh(a)	128	100	5	0.9773	20

Table 3: Multi-layer (tanh) Parameters and Results



(a) Cross Entropy Loss (b) Accuracy

Figure 3: Multi-layer Tanh Activation

2.2.2 Gradient Observation

From my multi-layer (tanh) neural network, I want to observe the gradient change after a forward and backward pass of a single pattern. I used the following formula to approximate the slope:

$$\frac{d}{dw}E^n(w) \approx \frac{E^n(w+\epsilon) - E^n(w-\epsilon)}{2\epsilon}$$

I set $\epsilon = 0.01$ and the difference between the approximated slope and generated gradient should not be greater than $O(\epsilon^2)$. Here are my approximations:

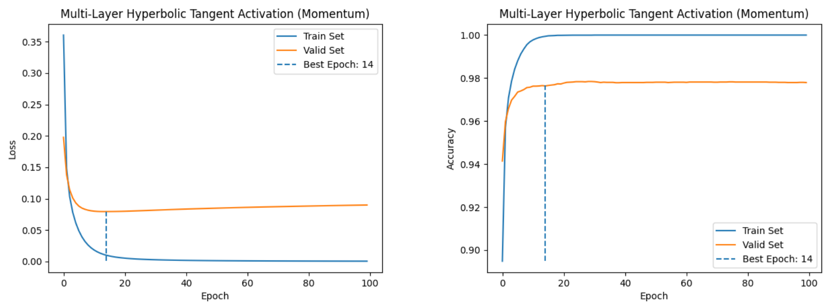
Type (Location)	Approximation	Backprop	Difference
Input to Hidden (0, 0)	0.001484	0.001462	2.216747e-05
Input to Hidden (0, 1)	0.002250	0.002254	3.658941e-06
Hidden to Output (0, 0)	0.048427	0.048427	1.871714e-10
Hidden to Output (0, 1)	0.004482	0.004482	1.064512e-10
Output Bias (128, 0)	0.900012	0.900013	1.199896e-06
Hidden Bias (784, 0)	0.003645	0.003633	1.161070e-05

2.2.3 Momentum Experiment

With the introduction of the momentum hyperparameter, we can converge on the gradient of the cross-entropy loss function faster. Momentum introduces the hyperparameter "γ", which essentially appends a small section of the previous weight onto the new weight. After setting a momentum parameter, we see that the model stops at an earlier epoch and has around the same accuracy.

α	$g(a)$	b_{size}	ϵ	ϵ_{stop}	γ	Test Accuracy	Stoppage
0.001	tanh(a)	128	100	5	0.9	0.9776	14

Table 4: Multi-layer (tanh + momentum) Parameters and Results



(a) Cross Entropy Loss (b) Accuracy

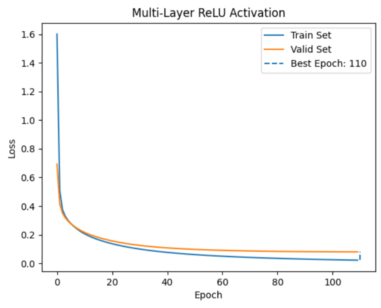
Figure 4: Multi-layer Tanh Activation (Momentum)

2.2.4 ReLU Activation

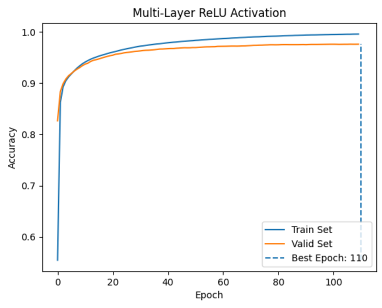
Another common activation function is the rectified linear unit ($ReLU(a) = max(0, a)$). Unlike the hyperbolic tangent function, the ReLU function does not allow negative values to be passed through. This activation function does a better job of singling out individual features but at the cost of over-shooting. The nature of this function required a lower learning step and a higher epoch, but in turn, it yielded better accuracy.

α	$g(a)$	b_{size}	ϵ	ϵ_{stop}	Test Accuracy	Stoppage
0.0001	relu(a)	128	110	5	0.9782	110

Table 5: Multi-layer (relu) Parameters and Results



(a) Cross Entropy Loss



(b) Accuracy

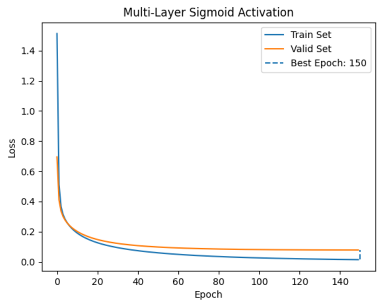
Figure 5: Multi-layer ReLU Activation

2.2.5 Sigmoid Activation

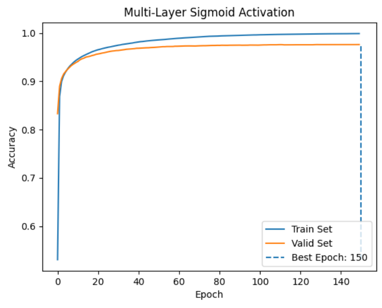
Similar to the hyperbolic tangent function, the sigmoid activation function ($sigmoid(a) = \frac{1}{1+e^{-a}}$) outputs a value in between the range [0, 1]. For this model, I substantially increased the epochs and lowered the learning rate. With these small adjustments, the sigmoid activation hidden layer model performed the best out of the other two activation functions.

α	$g(a)$	b_{size}	ϵ	ϵ_{stop}	Test Accuracy	Stoppage
0.0001	sigmoid(a)	128	150	5	0.9788	150

Table 6: Multi-layer (sigmoid) Parameters and Results



(a) Cross Entropy Loss



(b) Accuracy

Figure 6: Multi-layer Sigmoid Activation

3 Regularization

To avoid overfitting on the train set, I tried two types of regularization techniques on my multi-layer sigmoid activation model. I employed Lasso (L1) regularization and Ridge (L2) regularization. Both penalized my model through increased cross-entropy loss. The use of regularization did not have much of an impact on the accuracy of my model on the test set.

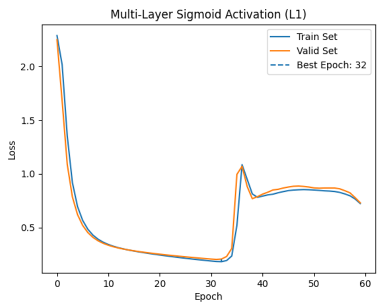
3.1 Lasso Regression

With L1 regression, I choose a small regularization term (λ_1) to adjust my weights. I penalized my model during every backward pass by adjusting the weights directly by the derivative of the absolute value function times the regularization term. Only a slight change in accuracy occurred with an earlier epoch stoppage. Here are the plots for loss and accuracy with lasso regression:

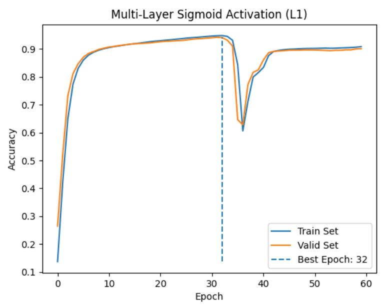
α	$g(a)$	b_{size}	ϵ	ϵ_{stop}	λ_1
0.0001	sigmoid(a)	128	60	3	0.0001

Test Accuracy	Stoppage
0.9634	32

Table 7: Multi-layer (sigmoid + L1) Parameters and Results



(a) Cross Entropy Loss



(b) Accuracy

Figure 7: Multi-layer Sigmoid Activation (L1)

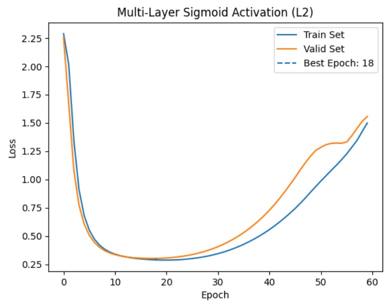
3.2 Ridge Regression

In ridge regression, this style of regularization penalizes large values. L2 regularization squares the weights and appends this to the total cross-entropy loss. I choose a small regularization term (λ_2) for L2 regularization. Similar to L1 regularization, stoppage happened earlier with a minor drop in accuracy. Here are the results of L2 regularization:

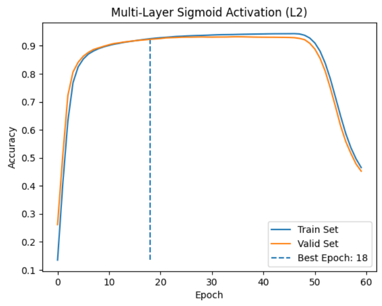
α	$g(a)$	b_{size}	ϵ	ϵ_{stop}	λ_2
0.0001	sigmoid(a)	128	60	3	0.0001

Test Accuracy	Stoppage
0.9641	18

Table 8: Multi-layer (sigmoid + L2) Parameters and Results



(a) Cross Entropy Loss



(b) Accuracy

Figure 8: Multi-layer Sigmoid Activation (L2)

4 Conclusion

In conclusion, the exploratory models I created yielded promising results, and the best of my models had an accuracy rate of 97.88%. Properties like learning rate, momentum, and regularization were important parameters in producing the best neural networks. A simple two-layered model employing the sigmoid activation function was the best in terms of accuracy and simplicity. Momentum and regularization had a marginal impact on accuracy while speeding up the training phase. Another approach I would have tried was to make a more complicated model with more hidden layers. I believe a three-layered model using the sigmoid function for the first hidden layer and ReLU activation for the second hidden layer would be promising.