Implementing a Multi-Layer Neural Network for MNIST Classification

Daniel Shi
Halıcıoğlu Data Science Institute
UC San Diego
La Jolla, CA 92092
dshi@ucsd.edu

Abstract

This report presents a series of experiments conducted on the MNIST dataset to evaluate different strategies in neural network design and optimization. Initially, a linear model with softmax regression was implemented, achieving a test accuracy of 92.51%. Subsequent experiments involved a network with a hidden layer, where gradient verification affirmed the accuracy of backpropagation. Further exploration included momentum-based SGD, L1 and L2 regularization, and various activation functions (Sigmoid, $\tanh$, and ReLU), with each modification analyzed for its impact on performance. The most notable result was from the ReLU activation function, which achieved the highest test accuracy and the lowest loss with 97.69%.

1 Data Loading

The dataset we loaded is the MNIST dataset, which is a widely used dataset consisting of handwritten digit images. Each image represents a digit from 0 to 9. The MNIST dataset contains a total of 70,000 images, divided into 60,000 training datapoints and 10,000 test datapoints. For the training data, it is split using an 80-20 ratio (80 percent for training, 20 percent for validation). According to this ratio, we expect to have 48,000 datapoints in X_{train} and 12,000 images in X_{val}.

Regarding the normalization procedure, we first calculated the mean of the given input 2D array, where N represents the number of examples and d represents the dimensions of the image. Then, we calculated the standard deviation of the input 2D array. Finally, we normalized the input array using the formula $(inp - mean)/std$, where inp is the input array. For any one training image, the mean is approximately 0, and the standard deviation is approximately 1.

For the labels of the Digit classes $t \in \{0, ..., 9\}$, To effectively use these in our softmax regression model, we employ one-hot encoding. We mapped the targets to a array of size 10 corresponding to the 10 digit classes and 1 at the actual label class and 0 else where.

2 Softmax Regression

In this experiment we trained a network without hidden layers using softmax regression from the input to output. The input to the network has 784 dimensions corresponding to the 28x28 pixel values of each MNIST image augmented with a bias term and the output has is 10 units corresponding to the 10 classes (0-9).

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

2.1 Softmax regression

The softmax regression is as following:

$$y_k^n = \frac{exp(a_k^n)}{\sum_{k'} exp(a_{k'}^n)} \tag{1}$$

$$a_k^n = w_k^T x^n \tag{2}$$

y_k^n here is the probability that data point n belongs to class k, a_k^n is the dot product of w_k and x^n.

2.2 Loss Function

The model's performance is evaluated using the cross-entropy loss function:

$$E = -\sum_n \sum_{k=1}^{c} t_k^n \ln y_k^n \tag{3}$$

This loss function is effective for classification problems with multiple classes, as it penalizes incorrect classifications heavily.

2.3 Gradient Computation

We utilize backpropagation to compute the gradient of the loss function with respect to the weights:

$$-\frac{\partial E^n(w)}{\partial w_{jk}} = (t_k^n - y_k^n) x_j^n \tag{4}$$

2.4 Training and Hyperparameter

We used a mini-batch Stochastic Gradient Decent algorithm to train our model.

Algorithm 1 Stochastic Gradient Descent

procedure STOCHASTIC GRADIENT DESCENT
  $w \leftarrow 0$
  for t = 1 to *M* **do** ▷ Here, t is one epoch.
    randomize the order of the indices into the training set
    for j = 1 to *N*, in steps of B **do** ▷ Here, N is number of examples and B is the batch size
      start = j
      end = j+B
      $w_{t+1} = w_t - learning\_rate \cdot \frac{1}{B} \cdot \sum_{n=start}^{end} \nabla E^n(w)$
  return w

We used a early stopping method to avoid overfitting. The method is to set a patience threshold of consecutive increase of the validation loss. If this is reached, the training will stop and return the model with best performance, which has the lowest validation loss.

The training hyperparameters we chose in our training is:

1. Activation: None
2. Learning rate: 0.05
3. Batch Size: 128
4. Early stop epoch: 3
5. Epochs to Run: 100

2.5 Results

The training early stopped at epoch 20. and After training, the model achieved a test accuracy of 92.51% and a test loss of 0.27.

The plots of the training loss and accuracy and validation loss and accuracy is provided below.

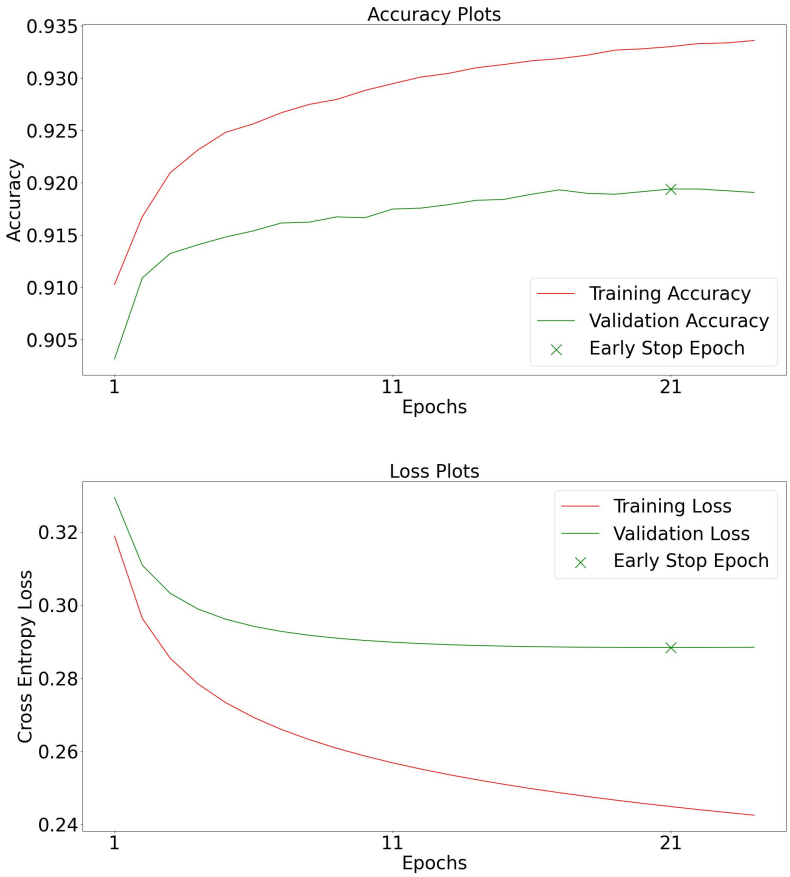

Figure 1: Softmax Plots

3 Gradient Experiments

In this experiment, we used a neural network with a hidden layer of 128 units using $\tanh$ activation to verify the correctness of our backpropagation gradient.

The numerical approximation equation is to approximate the gradient of the network is provided below:

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

where ϵ is a small constant, e.g. 10^{-2}. The difference of the gradients should be within big-O of ϵ^2, so if you used 10^{-2}, your gradients should agree within 10^{-4}.

The table below shows the weights selected and the corresponding gradient values computed via back-propagation alongside numerical approximation by inputting a random data sample. To verify the accuracy of the gradients, specific weights were chosen for testing: one bias weight from the output layer, one bias weight from the hidden layer, and four weights connecting the layers, two each from the hidden to output and input to hidden layers. The bias weights tested are the last ones in their respective layers.

weight types	Numerical Approximation	Backprop gradient	absolute difference
Output bias	-0.001305	-0.001309	4e-06
Hidden bias	-9.77e-05	0.000229	0.000327
Input to Hidden i	-6.390e-05	0.0002208	0.00028
Input to Hidden ii	-0.000137	0.000221	0.0003585
Hidden to Output i	-0.0021	-0.00208	2.08e-05
Hidden to Output ii	0.0204	0.02035	0.000102

Table 1: Numerical approximation experiment

From the table we can see the absolute difference between the numerical approximation gradient and the back propagation gradient is within 10^{-4}.

4 Momentum Experiments

In this experiment, we utilize mini-batch Stochastic Gradient Descent (SGD), adding momentum to our update rule to train our model with hidden layer of 128 units using $\tanh$ activation.

4.1 Momentum Update Rule

The momentum update rule is as following:

$$\Delta w^t = \gamma \cdot \Delta w^{t-1} + (1-\gamma) \cdot \alpha \frac{\partial E}{\partial w} \tag{5}$$

4.2 Training and Hyperparameter

We used the same mini-batch SGD algorithm to train our multi-layer neural network model. The training hyperparameters we chose in our training is:

1. Activation: $\tanh$
2. Learning rate: 0.1
3. Momentum gamma: 0.9
4. Batch Size: 128
5. Early stop epoch: 3
6. Epochs to Run: 100

4.3 Results

The training early stopped at epoch 31. and After training, the model achieved a test accuracy of 97.45% and a test loss of 0.08.

The plots of the training loss and accuracy and validation loss and accuracy is provided below. The plots of the training loss and accuracy and validation loss and accuracy is shown below.

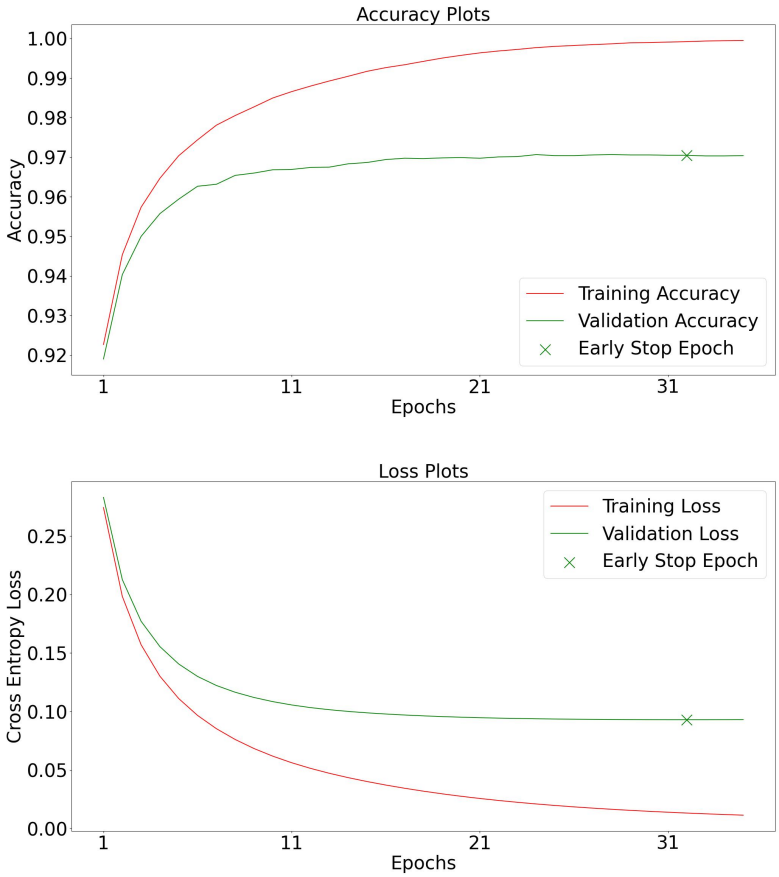

Figure 2: Momentum Plots

5 Regularization Experiments

Regularization is a crucial technique to prevent overfitting, where the model performs well on the training data but poorly on unseen data. Regularization methods modify the learning process to reduce overfitting, improving the model's generalization capabilities. In our implementation, we add a regularization term to our gradient.

For L2 penalty, the term added to the gradient is,

$$\frac{\lambda}{2}\frac{\partial}{\partial w}w^2 = \lambda w \tag{6}$$

And for L1 penalty,

$$\frac{\lambda}{2}\frac{\partial}{\partial w}|w| = \lambda \cdot sign(w) \tag{7}$$

Where λ is the regularization constant for both terms.

5.1 L2 Regularization

Training hyperparameters:

1. Activation: $\tanh$
2. Learning rate: 0.05
3. Momentum gamma: 0.9

4. Batch Size: 128

5. Early stop epoch: 3

6. Epochs to Run: 100

We chose $\lambda = 0.01$ as our L2 regularization constant as it can show the regularization process more clearly.

The result of the training procedure is that it early stopped at epoch 5 with a test accuracy of 95.79% with a test loss 0.155. We can see in the plot that as the training goes on, the training loss keeps decreasing and the validation loss first goes down and at a point, it starts increasing because of the increasing complexity of the weights of the neural network and that point is where we early stopped at.

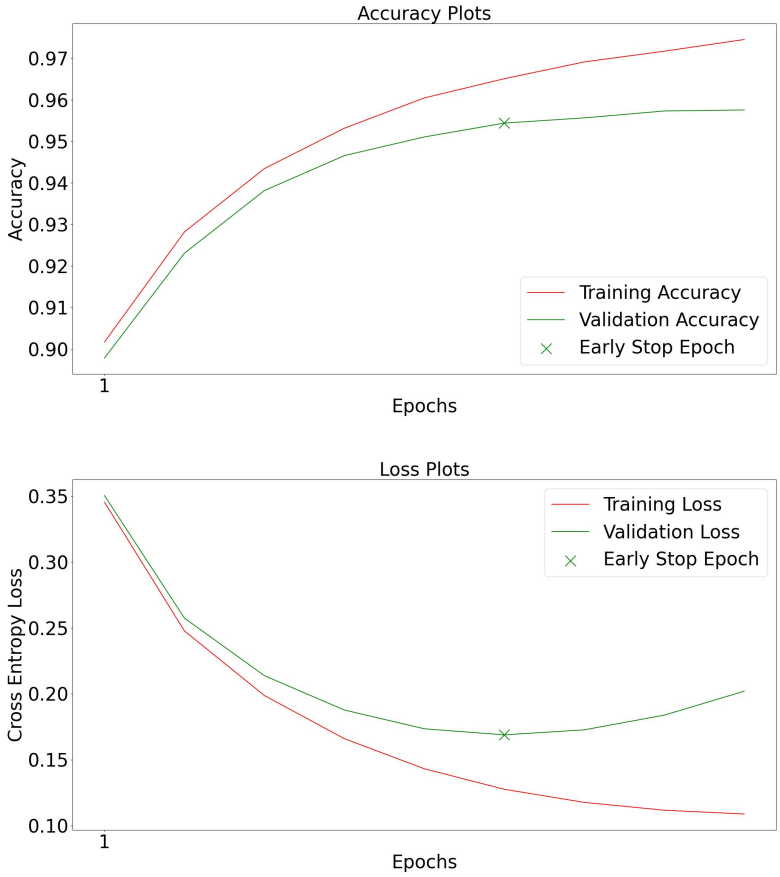

Figure 3: $\lambda = 0.01$

5.2 Comparison of L1 and L2 Regularization

We chose $\lambda = 0.01$ as our L1 regularization constant and everything else the same to compare with L2 regularization. The test accuracy is 95.18% and test loss is 0.1649. The test accuracy is a bit smaller than that of L2: 95.79%, and the test loss is bigger than L2: 0.155. According my prior knowledge, L1 penalty might lead to sparse weights, making some of the weights 0, since its gradient disregards the magnitude of the weights and penalize harder on small values compared to that of L2 penalty.

6 Activation Experiments

6.1 Sigmoid

The activation function of Sigmoid is:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

Training hyperparameters:

1. Activation: sigmoid
2. Learning rate: 0.5
3. Momentum gamma: 0.9
4. Batch Size: 128
5. Early stop epoch: 3
6. Epochs to Run: 100

Result: Early stopped at epoch 27, Test Accuracy: 97.43 % Test Loss: 0.0823. This has similar performance with $\tanh$ activation for both having 97% test accuracy and 0.08 test loss.

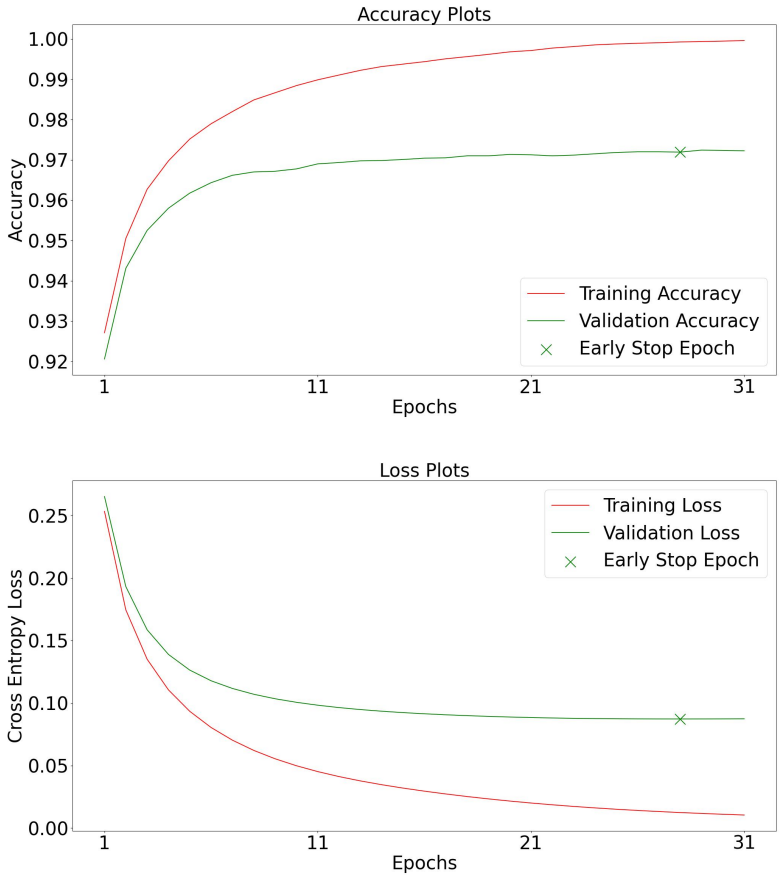

Figure 4: Sigmoid plots

6.2 ReLU

The activation function of ReLU is:

$$f(x) = \max(0, x) \tag{9}$$

Training hyperparameters:

1. Activation: ReLU
2. Learning rate: 0.1
3. Momentum gamma: 0.9
4. Batch Size: 128
5. Early stop epoch: 3
6. Epochs to Run: 100

Result: Early stopped at epoch 17, Test Accuracy: 97.69 % Test Loss: 0.0794. ReLU has the best performance among all the other activation functions with a slightly better test accuracy and test loss.

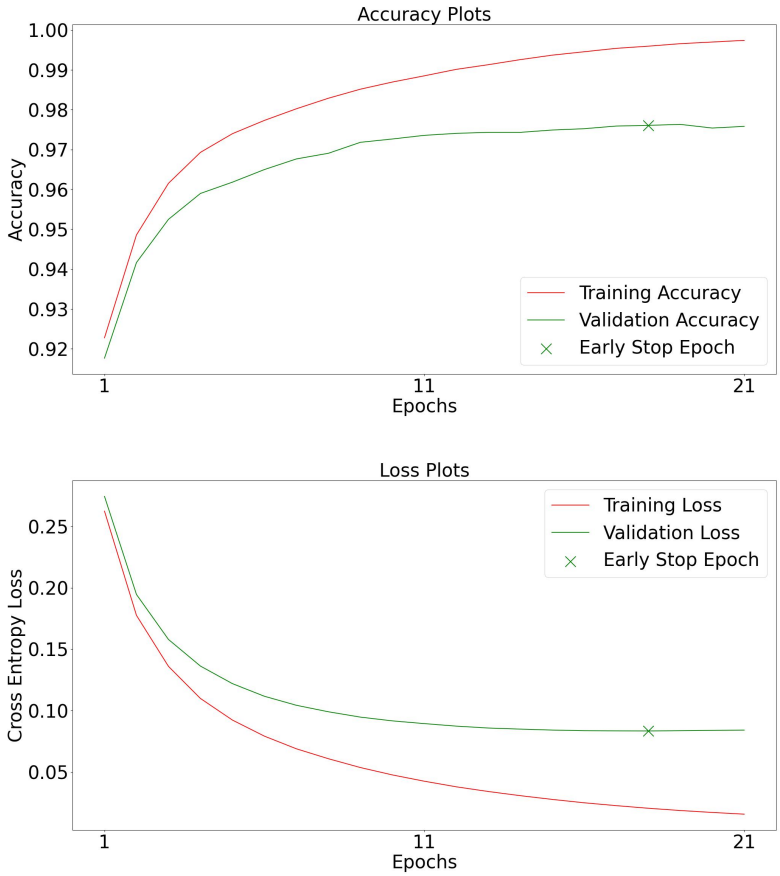

Figure 5: ReLU plots

7 Team Contributions

I grouped up with one teammate before starting the assignment, but my teammate dropped the course after we grouped up. All the code and report are written by myself.

References

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278-2324.