MNIST Classifier Neural Network from Scratch

Philip Paterson

March 2024

1 Introduction - Problem Statement

In Programming Assignment 2, we program a deep neural network, with two hidden layers, to perform multi-class classification to classify images of single hand-written digits (0-9) from the MNIST dataset.

In this report, we will first describe the theories used to write this program, mainly the theories of implementing multi-class classification using a neural network with two hidden layers, stochastic gradient descent, and how parameters are updated from forward and backward propagation.

2 Model Description

The model performs multi-class classification utilizing a neural network. The architecture of the neural network can be described as having one input layer, $\mathbf{x}$, with two hidden layers H^1 and H^2, and an output layer Y. The activation function, g, between the last hidden layer H^2 and the output layer Y is the softmax activation function, and the activation functions between the input layer, H^1, and H^2, are the ReLU function ϕ.

In this section we will describe the loss function used for this model. Afterwards, we will go into stochastic gradient descent, forward propagation, backward propagation, and how the parameters are updated. The following derivations frequently reference Dr. Qiang Ji's lecture material for the Rensselaer Polytechnic Institute Course ECSE 4850.

2.1 Loss Function

Given $\mathbf{D} = \{\mathbf{x}[m], \mathbf{y}[m]\}, m = 1, 2, ..., M$; $\Theta = \begin{bmatrix} \Theta^1 \\ \Theta^2 \\ \dots \\ \Theta^{L+1} \end{bmatrix}$, where

$\Theta^l = \begin{bmatrix} \Theta^l_1 & \Theta^l_2 & ... & \Theta^l_K \end{bmatrix}$ and $\Theta^l_k = \begin{bmatrix} \mathbf{W}^l_k \\ \mathbf{W}^l_{0,k} \end{bmatrix}$; $\mathbf{W}^l = \begin{bmatrix} \mathbf{W}^l_1 & \mathbf{W}^l_2 & ... & \mathbf{W}^l_K \end{bmatrix}$ and

$\mathbf{W}_0^l = \begin{bmatrix}\mathbf{W}_{0,1}^l & \mathbf{W}_{0,2}^l & ... & \mathbf{W}_{0,k}^l\end{bmatrix}$; $\mathbf{X} = \begin{bmatrix}\mathbf{x} \\ 1\end{bmatrix}$; $\mathbf{y}$ is 1-of-K encoded, where K is the number of classes (in this case, K = 10).

The type of the loss function we will use for multiclass classification L is the Negative Log Likelihood (NLL) loss function.

$$\begin{aligned} L(\mathbf{D} : \Theta) &= -\frac{1}{M}\sum_{m=1}^{M} \log p(\mathbf{y}[m] | \mathbf{X}[m], \Theta) \\ &= -\frac{1}{M}\sum_{m=1}^{M} \log \left(\prod_{k=1}^{K} [p(\mathbf{y}[m][k] = 1 | \mathbf{X}[m], \Theta)]^{\mathbf{y}[m][k]} \right) \\ &= -\frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K} \mathbf{y}[m][k] \log p(y[m][k] = 1 | \mathbf{X}[m], \Theta) \end{aligned}$$

Additionally, we define the predicition $\hat{\mathbf{y}} = p(y[m][k] = 1 | \mathbf{X}[m], \Theta)$.

Thus, because we are using a multiclass sigmoid function, or softmax function (σ_M), use the softmax function for $p(y[m][k] = 1 | \mathbf{X}[m], \Theta)$, where

$$\sigma_M(\hat{\mathbf{y}}) = \frac{\exp(\hat{\mathbf{y}}_k)}{\sum_{j=1}^{K} \exp(\hat{\mathbf{y}}_j)}$$

So the loss function we utilize, L, is...

$$L(\mathbf{D} : \Theta) = -\frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K} \mathbf{y}[m][k] \log \sigma_M(\hat{\mathbf{y}}[m][k])$$

where

$$l(y[m], \hat{\mathbf{y}}[m]) = -\sum_{k=1}^{K} \mathbf{y}[m][k] \log(\hat{y}[m][k])$$

2.2 Stochastic Gradient Descent

The ideal parameters, represented by Θ^*, are found by minimizing the negative log-likelihood loss function, where $\Theta^* = \underset{\Theta}{\text{argmin}} L_{NLL}(\mathbf{D} : \Theta)$. However, because there is no closed-form solution, Θ^* must be estimated through an iterative process, where for the t^{th} epoch,

$$\Theta^t = \Theta^{t-1} - \eta \frac{\partial L_{NLL}(\mathbf{D} : \Theta)}{\partial \Theta}$$

Furthermore, for added efficiency because of just how large our data $\mathbf{D}$ is, we perform stochastic gradient descent, or SGD, to approximately compute the gradient of the the loss function.

This is achieved by randomly dividing $\mathbf{D}$ into $K = 1000$ mini-batches of $S = 50$ size, where D $= \begin{bmatrix} D_1 & D_2 & ... & D_K \end{bmatrix}$. Thus, at each iteration k, we randomly select a mini-batch D_k and calculate the average gradients using the formula, utilizing the loss function described in section 2.1:

$$\frac{\partial L_{NLL}(\mathbf{D} : \Theta)}{\partial \Theta} = \frac{1}{S} \sum_{\mathbf{x}[m], y[m] \in D_k} \frac{\partial l(\mathbf{x}[m], y[m], \Theta)}{\partial \Theta}$$

Note: **For my hyper-parameters, my learning rate is $\eta = 0.13$ and my maximum number of epochs is 2**

Now we must tackle computing the gradients necessary to compute the ideal parameters Θ^* by performing backward propagation.

2.3 Backward Propagation

The backward propagation algorithm can be summarized to be performed in three steps:

1. **Forward Pass:** Initialize the weights Θ and propagation forward by running the neural network to calculate $\hat{y}$.
2. **Backward Pass:** First compute the gradient of the output, $\nabla \hat{y} = \nabla_{\hat{y}} l(y[m], \hat{y}[m])$, and pass $\nabla \hat{y}$ backwards by recursively computing the gradients of the weights, $\nabla \mathbf{W}^l$ and $\nabla \mathbf{W}_0^l$ for each layer l.
3. **Update the Weights:** Utilize the approximated gradients to update the weights for each layer of the neural network.

2.3.1 Forward Pass

Given $\mathbf{D} = \{\mathbf{x}[m], \mathbf{y}[m]\}, m = 1, 2, ..., M$; $\Theta = \begin{bmatrix} \Theta^1 \\ \Theta^2 \\ \cdots \\ \Theta^{L+1} \end{bmatrix}$, where

$\Theta^l = \begin{bmatrix} \Theta_1^l & \Theta_2^l & ... & \Theta_K^l \end{bmatrix}$ and $\Theta_k^l = \begin{bmatrix} \mathbf{W}_k^l \\ \mathbf{W}_{0,k}^l \end{bmatrix}$; $\mathbf{W}^l = \begin{bmatrix} \mathbf{W}_1^l & \mathbf{W}_2^l & ... & \mathbf{W}_K^l \end{bmatrix}$ and $\mathbf{W}_0^l = \begin{bmatrix} \mathbf{W}_{0,1}^l & \mathbf{W}_{0,2}^l & ... & \mathbf{W}_{0,k}^l \end{bmatrix}$; $\mathbf{X} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$; $\mathbf{y}$ is 1-of-K encoded, where K is the number of classes (in this case, K = 10).

The calculation of the following layers can be computed as follows:

$$\begin{aligned} \mathbf{H}^1[m] &= \phi((\mathbf{W}^1)^T x[m] + \mathbf{W}_0^1) \\ \mathbf{H}^2[m] &= \phi((\mathbf{W}^2)^T x[m] + \mathbf{W}_0^2) \\ \hat{\mathbf{y}}[m] &= \phi((\mathbf{W}^3)^T x[m] + \mathbf{W}_0^3) \end{aligned}$$

2.3.2 Backward Pass

For any layer at index l (where the output layer, $\hat{\mathbf{y}}[m]$ is at index $L+1$), the following gradients can be computed as follows, and one can recurse backwards using the previous layers gradient for the other previous gradients.

Note: $z(m) = (W^l)^T H^{(m)} + W_0^l$

$$\nabla W^l[m] = \frac{\partial \hat{y}[m]}{\partial W^l} \nabla \hat{y}[m] = \frac{\partial z(m)}{\partial W^l} \frac{\partial g(z(m))}{\partial z(m)} \nabla \hat{y}[m]$$

$$\nabla W_0^l[m] = \frac{\partial \hat{y}[m]}{\partial W_0^l} \nabla \hat{y}[m] = \frac{\partial z(m)}{\partial W_0^l} \frac{\partial g(z(m))}{\partial z(m)} \nabla \hat{y}[m]$$

$$\nabla H^{l-1}[m] = \frac{\partial \hat{y}[m]}{\partial H^{l-1}} \nabla \hat{y}[m] = \frac{\partial z(m)}{\partial H^{l-1}} \frac{\partial g(z(m))}{\partial z(m)} \nabla \hat{y}[m]$$

Thus, for the ouput layer where $l = L+1$:
$z(m) = (W^{L+1})^T H^{(m)} + W_0^{L+1}$

$$\nabla W^{L+1}[m] = \frac{\partial \hat{y}[m]}{\partial W^{L+1}} \nabla \hat{y}[m] = \frac{\partial z(m)}{\partial W^{L+1}} \frac{\partial g(z(m))}{\partial z(m)} \nabla \hat{y}[m]$$

$$\nabla W_0^{L+1}[m] = \frac{\partial \hat{y}[m]}{\partial W_0^{L+1}} \nabla \hat{y}[m] = \frac{\partial z(m)}{\partial W_0^{L+1}} \frac{\partial g(z(m))}{\partial z(m)} \nabla \hat{y}[m]$$

$$\nabla H^{L}[m] = \frac{\partial \hat{y}[m]}{\partial H^{L}} \nabla \hat{y}[m] = \frac{\partial z(m)}{\partial H^{L}} \frac{\partial g(z(m))}{\partial z(m)} \nabla \hat{y}[m]$$

and the function g is the softmax output function:

$$g(z[m]) = \sigma_M(z[m]) = \begin{pmatrix} \sigma_M(z[m][1]) \\ \sigma_M(z[m][2]) \\ \vdots \\ \sigma_M(z[m][K]) \end{pmatrix}$$

To calculate the gradient of the bias:

$$\nabla W_0^{L+1}[m] = \frac{\partial z(m)}{\partial W_0^{L+1}} \frac{\partial \sigma_M(z(m))}{\partial z(m)} \nabla \hat{y}[m] = \frac{\partial \sigma_M(z(m))}{\partial z(m)} \nabla \hat{y}[m]$$

as $\frac{\partial z(m)}{\partial W_0^{L+1}}$ can be reduced to 1 in this case.
The gradient of the softmax is as follows:

$$\left(\frac{\partial \sigma_M(z[m])}{\partial z[m]}\right)^{K \times K} = \begin{bmatrix} \frac{\partial \sigma_M(z[m][1])}{\partial z[m]} & \frac{\partial \sigma_M(z[m][2])}{\partial z[m]} & \cdots & \frac{\partial \sigma_M(z[m][K])}{\partial z[m]} \end{bmatrix}$$

$$\text{where } \frac{\partial \sigma_M(z[m][k])}{\partial z[m]} = \begin{bmatrix} \frac{\partial \sigma_M(z[m][k])}{\partial z[m][1]} \\ \frac{\partial \sigma_M(z[m][k])}{\partial z[m][2]} \\ \vdots \\ \frac{\partial \sigma_M(z[m][k])}{\partial z[m][K]} \end{bmatrix}$$

$$\text{where } \frac{\partial \sigma_M(z[m][k])}{\partial z[m][i]} = \begin{cases} \sigma_M(z[m][i])(1-\sigma_M(z[m][i])) & \text{if } k = i \\ -\sigma_M(z[m][k])\sigma_M(z[m][i]) & \text{else} \end{cases}$$

Thus, ∇W^{L+1} and $\nabla H^L[m]$ can be rewritten in terms of ∇W_0^{L+1} as follows:

$$\nabla W^{L+1}[m] = \frac{\partial z(m)}{\partial W^{L+1}} \frac{\partial g(z(m))}{\partial z(m)} \nabla \hat{y}[m] = \frac{\partial z(m)}{\partial W^{L+1}} \nabla W_0^{L+1}$$

$$\nabla H^L[m] = \frac{\partial z(m)}{\partial H^L} \frac{\partial g(z(m))}{\partial z(m)} \nabla \hat{y}[m] = \frac{\partial z(m)}{\partial H^L} \nabla W_0^{L+1}$$

where

$$\left(\frac{\partial z[m]}{\partial W^{L+1}}\right)^{N_L \times K \times K} = \begin{bmatrix} \frac{\partial z[m][1]}{\partial W^{L+1}} & \frac{\partial z[m][2]}{\partial W^{L+1}} & \cdots & \frac{\partial z[m][K]}{\partial W^{L+1}} \end{bmatrix}$$

$$\text{where } \frac{\partial z[m][k]}{\partial W^{L+1}} = \begin{bmatrix} \frac{\partial z[m][k]}{\partial W^{L+1}[][1]} & \frac{\partial z[m][k]}{\partial W^{L+1}[][2]} & \cdots & \frac{\partial z[m][k]}{\partial W^{L+1}[][K]} \end{bmatrix}$$

$$\text{where } \frac{\partial z[m][k]}{\partial W^{L+1}[][i]} = \frac{\partial((W_k^{L+1})^T H^L(m) + W_0^{L+1}[k])}{\partial W^{L+1}[][i]} = \begin{cases} H(m) & \text{if } k = i \\ 0 & \text{else} \end{cases}$$

.

For the two hidden layers, follow a similar process except for the derivative of the function g, which would be the ReLU function in that case, where

$$\phi(z[m]) = \begin{bmatrix} \phi(z[m][1]) \\ \phi(z[m][2]) \\ \vdots \\ \phi(z[m][N_l]) \end{bmatrix}$$

and $\frac{\partial \phi(z[m][i])}{\partial z[m]} = \begin{bmatrix} \frac{\partial \phi(z[m][i])}{\partial z[m][1]} \\ \frac{\partial \phi(z[m][i])}{\partial z[m][2]} \\ \vdots \\ \frac{\partial \phi(z[m][i])}{\partial z[m][N_l]} \end{bmatrix}$

$$\frac{\partial \phi(z[m][i])}{\partial z[m][j]} = \begin{cases} 1 & \text{if } i = j \text{ and } z[m][i] > 0 \\ 0 & \text{else if } i \neq j \text{ and } z[m][i] \neq 0 \\ c & \text{else if } z[m][j] = 0 \end{cases}$$

In this case, I set my $c = 0$.

2.3.3 Update the Weights

After each backward pass, update the weights for each layer l as follows:

$$W^l[t] = W^l[t-1] - \eta(\nabla W^l[m])$$
$$W_0^l[t] = W_0^l[t-1] - \eta_0(\nabla W_0^l[m])$$

And repeat the cycle of forward pass, backward pass, and weight updates for every batch.

3 Experimental Setup

The training and testing data from the MNIST dataset, where each image is a 28x28 image of a handwritten dataset, is split where 50,000 images are in the training dataset and 4,000 images are in the testing dataset.

The training dataset is split into 1000 batches of size $S = 50$.

The weight matrices of the model were initialized using Xavier Weight Initialization (described in this article: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/) where the range of the weight values of $\mathbf{W}_l$ are a uniform distribution between $-\frac{1}{sqrt(N_{l-1})}$ and $\frac{1}{sqrt(N_{l-1})}$, where N_{l-1} is the number of nodes in the previous layer $l-1$.

The learning rate was set equal to 0.13 to promote faster convergence and after testing which learning rate best achieved a 97% testing accuracy over the fewest number of epochs.

4 Experimental Results

Final average classifcation error on testing dataset: 97.2%

The figures 1-5 can be seen below.

5 Analysis

Even though I achieved 97% accuracy, my model leaves much to be desired. From my experimental results figures it can be seen that the accuracy was not accurately calculated as there was not enough precision for the accuracy calculations. Additionally, it seems that 4s and 9s are often confused, which makes sense because humans can often confuse them and because of their similar shapes. In general, it seems that digits with similar shapes were confused.

6 Conclusion

In this report, I applied the theories of multi-class classification for classifying MNIST digits 0-9 using a neural network with two hidden layers. A good future direction for this assignment would be to utilize a database to upload and load

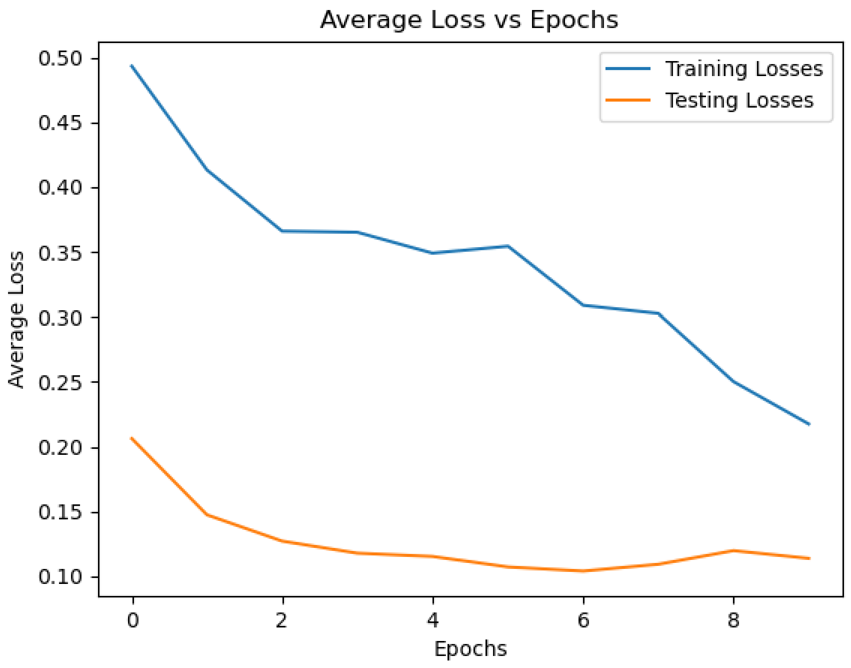


Figure 1: Average Loss vs Epochs

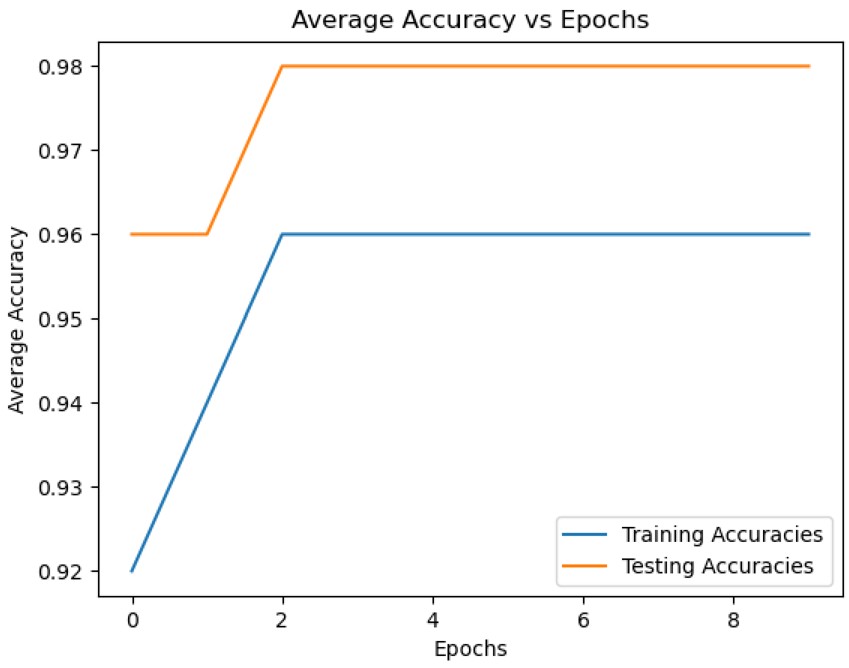


Figure 2: Average Accuracy vs Epochs

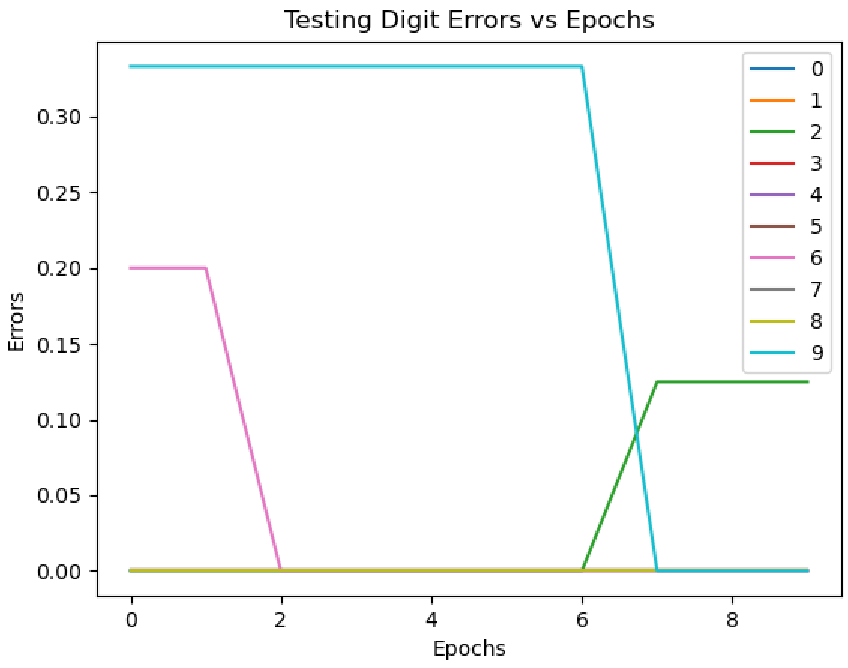


Figure 3: Testing Digit Errors vs Epochs

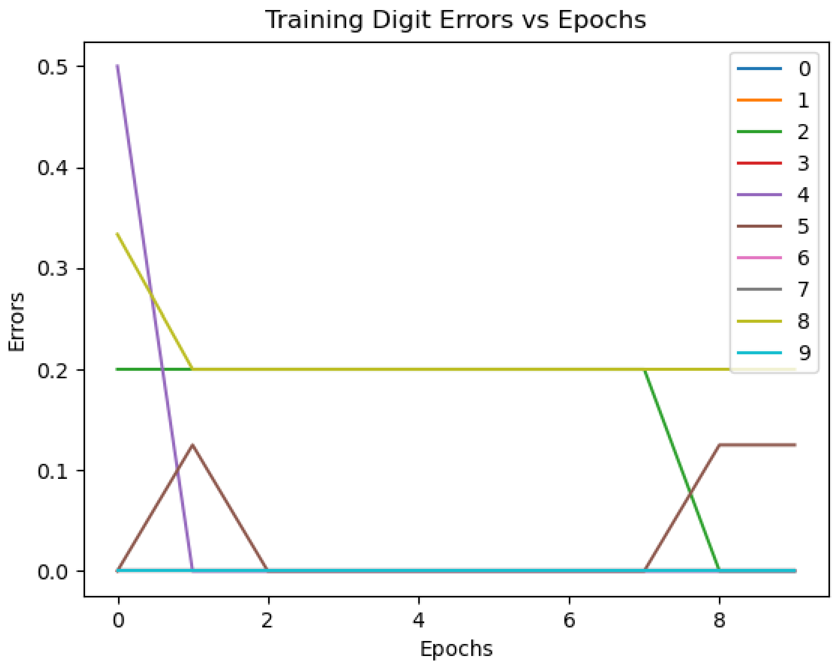
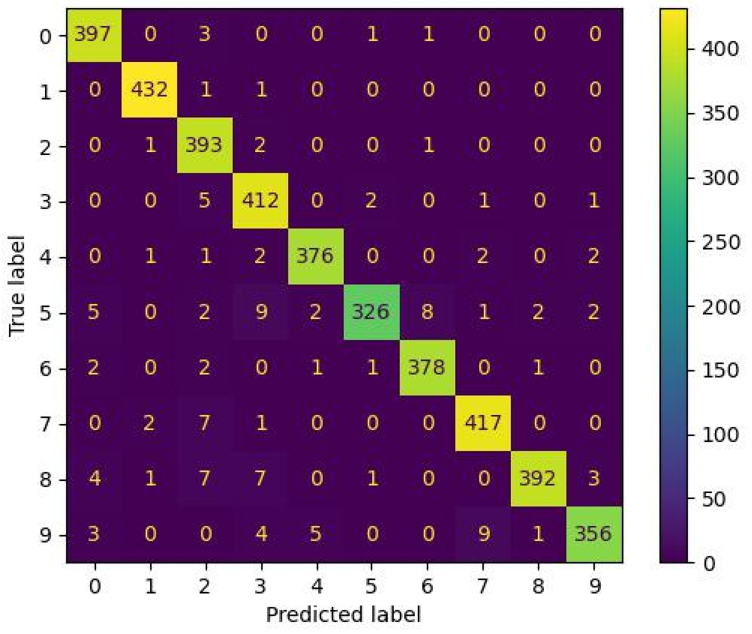


Figure 4: Training Digit Errors vs Epochs



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	397	0	3	0	0	1	1	0	0	0
1	0	432	1	1	0	0	0	0	0	0
2	0	1	393	2	0	0	1	0	0	0
3	0	0	5	412	0	2	0	1	0	1
4	0	1	1	2	376	0	0	2	0	2
5	5	0	2	9	2	326	8	1	2	2
6	2	0	2	0	1	1	378	0	1	0
7	0	2	7	1	0	0	0	417	0	0
8	4	1	7	7	0	1	0	0	392	3
9	3	0	0	4	5	0	0	9	1	356

Figure 5: Confusion Matrix for Class Digits 1-5

the data, make the program more modular to be able to make the program architecture more flexible. Ideally I would be able to expand the architecture based on user-inputted parameters.