Basil Lone

CS70

Programming Assignment 4: Neural Network

Project Technical Report

Implementation

To implement my neural network, I filled in the starter code for the Linear, ReLU, and MSELoss classes. For the linear layer class, I wrote the forward and backward methods to carry out the correct matrix multiplications and return and store the correct matrices. The forward layer carries out matrix multiplication of the input with the stored weights to obtain the input values of the next layer in the network. The backward function stores the partial derivative of the loss with respect to the weights and returns the partial derivative of the loss with respect to the input values of the forward function of the current layer.

In the ReLU layer class, I completed the forward and backward functions. In the forward function, first, the input to the layer is stored as an instance variable. Then, I use nested for loops to iterate over every element in the input matrix and set the value of the corresponding element in the output matrix to the maximum value from 0 and the input element. This carries out the ReLU function so then the output matrix can be returned.

For the backward function in the ReLU layer class, I use nested for loops to iterate over every element in the input matrix of the forward function of this layer. I check if each element is larger than 0 using an if statement within the nested for loops. If the current element is larger than 0, the corresponding element in the output matrix is set to the same value as the corresponding element in the input matrix of the backward function of this layer (this matrix gives the partial derivative of the loss with respect to the output of the forward function of this layer). If the element is less than or equal to 0, the corresponding value in the output matrix is left as 0. The resulting output matrix is the partial derivative of the loss with respect to the input of the forward function of this layer.

In the MSELoss layer class, I wrote the forward and backward methods. The forward function takes the final prediction calculated by the forward functions of the neural network and the true values to calculate the Mean Squared Error Loss. The difference between the prediction matrix and true value matrix is stored as an instance variable to be used in the backward function.

The backward function of the MSELoss layer class calculates the partial derivative of the loss with respect to the prediction and returns this matrix.

The final testing accuracy is calculated by running the backward functions of each layer after sequentially running each forward layer to determine predictions.

The network architecture is setup by using the Network class. This takes a network architecture specified by tuples as an input and creates the specified layers and appends them to a list to represent the neural network as an ordered list of linear and ReLU layers.

The forward function of the Network class runs all the forward functions for every layer in order, and the backward function runs the backward functions for every layer in the reverse order of the list of layers. The forward function generate and returns the prediction of the neural network, and the backward function generate and returns the partial derivative of the loss with respect to the input of the first layer of the network (this is run after generating a prediction from forward).

To complete the classifier, I wrote the function to carry out one hot encoding on the label data to make it suitable for use in the Classifier class. This was achieved by taking a vector of labels as input and creating a new row of 0s and 1s for each label in the vector.

To complete the Classifier class, I implemented gradient descent using the code from the Regressor class to carry out gradient descent. One of the variables had to be changed to use the one hot encoded version of the label data instead of the raw vector of labels.

Experiment Setup

The goal of the final experiment was to correctly identify handwritten numbers using a network architecture specified in the starter code.

The experiment was setup with the following values:

Learning rate = 0.01
Batch size = 32
Maximum epoch = 200

These values are from the starter code provided. I have chosen to stick with these and not change these as the testing accuracy I achieved with these values was above 0.8.

Network architecture:

Layer 1:
Linear
Input size: 784
Output size: 256

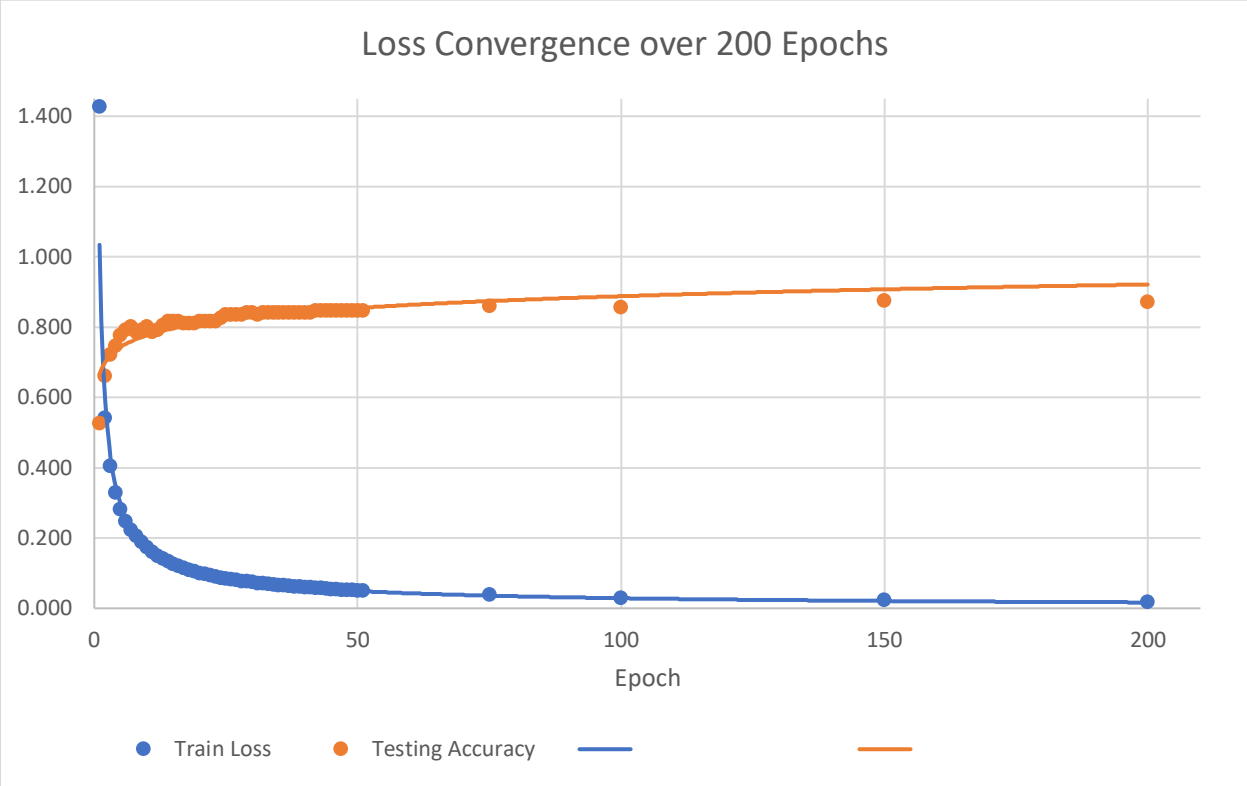
Layer 2:
ReLU
Input size: 256

Output size: 256

Layer 3:
Linear
Input size: 256
Output size: 10

The training and testing data was provided in the files from Canvas.

Network Performance



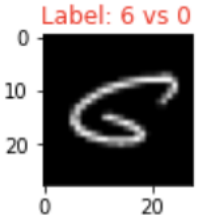
The plot above shows the improvement in the performance of the neural network with each epoch.

This shows that the network is performing as expected and achieves an acceptable testing accuracy. The final testing accuracy was reported as 0.87, indicating strong performance on the test data.

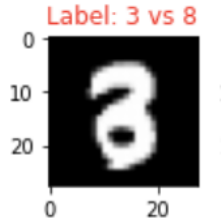
Failure cases

Three failure cases are shown below:

This shows messy handwriting of a 6. This would be hard to identify as the number 6, even for a human like me. To account for this, we would need more training data in this hand writing style so the neural network is more prepared to recognize this style of writing the numbers.



This shows the number 3. This, again, depicts hand writing that is hard to understand, even for a human like me. This unique style of writing would need to be present more in the training data to be able to identify it correctly in the test data.



This picture shows a relatively neat 7, so this indicates that there is room for improvement in the neural network as this seems simple enough to be identified by the model. This may indicate that we need to do some parameter tuning and maybe adjust the network architecture.