Vaani Gupta and Ryan Yong
COSC 70
Final Programming Assignment: Neural Network Building

PA Technical Report

Methodology Overview

The purpose of this project is to implement a neural network to perform learning on hand-written digits to classify them. To accomplish this purpose, we utilize linear algebra, least squares, chain rule, gradient descent, and nonlinear optimization. We use the MNIST dataset of hand-written digits for this project.

We began by using least squares in order to classify subsets in the MNIST dataset in the appetizer task. Next, we began the entree tasks by looking at the spec sheet listed associated with each task. For task 1, we coded the forward and backward methods for the LinearLayer class. We then checked the validity of our code with checkpoint 1 to ensure that our error converges to 0. For task 2, we coded the forward and backward methods for the ReLU class for non-linear layers. We then checked the validity of our code with checkpoint 2 to ensure that our error converged to 0. For task 3, we coded the forward and backward for the MSELoss class for the MSE loss function. We did not do a check on task 3 and went to task 4 directly. For task 4, we coded the Network class, including the constructor, forward, and backward methods. We then checked the validity of our code from task 3 and 4 with checkpoint 3 by checking that the training loss was decreasing and accuracy was increasing with each batch of data. For task 5, we coded the classifier to be used with gradient descent in which each label is associated with its respective index in the row of the class matrix. We then checked the functionality of our project with the evaluation by checking that the training loss was decreasing and testing accuracy was generally increasing with each batch of data.

Throughout the assignment, trial and error of our code allowed us to properly debug errors in our code by checking at the individual checkpoints provided.

Code Implementation

For the appetizer, we implemented a least squares solution to predict the labels in the images. To do this, we constructed a matrix consisting of a column for the bias term and then the classification columns for each label between 0 and 9. Specifically, if the train label is value y, then the y^{th} column for that row is 1 but the other remaining columns have value of -1. Hence, using the np.linalg.lstsq function and with the input X matrix and the newly constructed matrix described above, the parameters can be solved for. Using the least squares solution and given test data, we can use the argmax class predictor to predict the label for that data.

We constructed a neural network made up of linear and the ReLU nonlinear layers. Hence, to implement this network, we created a linear layer class, non linear layer class, a loss function class, and an overall network class. The linear layer class stored the weights (an m x n matrix), gradient of the weights, and the input X, while the nonlinear class solely stored the input matrix X. Both classes implemented forward and backward functions. For a linear layer, the forward function multiplied the input X with the weights matrix, while the nonlinear layer's forward function assigned each value in the input X with that value or with 0 depending on whether or not it's positive or non-negative. For both layers, their backward functions take the partial loss with respect to the layer's output Y as an input and output the partial loss with respect to the layer's input X.

The loss function class stores the difference between the predicted Y output value calculated by the network and the actual Y output noted in the data sample. It also has a forward function that calculates loss using the formula $\frac{1}{\#\, of\, samples}(Y^{pred} - Y^{truth})^2$. This class' backwards function finds the partial loss with respect to the layer's calculated predicted Y.

Finally, the Network class actually constructs the neural network using the linear and nonlinear classes. To initialize an object of this class, it takes a list of the type of layer and the necessary parameters (such as size of weights matrix for a linear layer). Hence, using this parameter, all of the layers noted in the list must be constructed using the linear and nonlinear classes. Often, the neural network has an alternating pattern between linear layer and nonlinear layer. The Network class also has a forward and backward function. The forward function takes an input X and then propagates this input throughout the entire network. Hence, the output from the forward function of a single layer becomes the input for the forward function of the next layer. Similarly, the backwards function takes the partial loss with respect to the calculated predicted Y and uses each layer's backwards function to output the partial loss with respect to the input X.

The actual Y values from the data samples must be processed to a matrix. Hence, the function One_Hot_Encode initializes a matrix of zeros with the number of rows equal to the number of samples and the number of columns equal to the number of classes (which was 10). Then, with a list of digits as a parameter, the function took each digit, denoted by i, and for its respective row, changed the 0 in the i^{th} column to 1.

Lastly, the Train Epoch function in the Classifier class trained the neural network using gradient descent. For each data sample, it calculated the predicted Y output using the Network class' forward function and calculated the loss between this Y and the corresponding one hot encode vector produced by the One_Hot_Encode function. Then, the partial loss with respect to the input X was found using the Network class' backward function. Then, for all linear layers in the network, gradient descent was conducted to find the parameters that would minimize the loss.

Experiment Setup and Results

For this experiment, we used a training data set of 1000 images, where each image was processed as a 28 x 28 image that contains labels that are a handwritten digit between 0 and 9. This training set is stored in a 1000 x 784 matrix. There is a testing data set of 200 images, where these images were stored in a 200 x 784 matrix.

For the appetizer, we trained and tested the data using a least squares solution/implementation to predict the test label. With this method, while the training accuracy was 99.8%, the testing accuracy was only 60%.

Furthermore, using the training data set, we constructed a neural network that contains 2 linear layers and one nonlinear one in an alternating pattern. Specifically, the nonlinear activation layer used the ReLU function. When a value x is inputted into this ReLU function, it either outputs x if it is greater than or equal to 0, or it outputs 0 if x is negative. This neural network was trained using gradient descent, where the step size, or learning rate, was 0.001. Once the network has been trained, the network is tested using the testing data set and its accuracy is calculated.

After constructing our neutral network, we ran it on our testing data set and found that as the number of batch sizes, or epochs, increases, the loss decreases and the accuracy increases. We got an overall accuracy of 86% from this neural network.

Network Efficacy

We measured network efficacy by examining the loss convergence rate (Figure 1.1), the change in loss with each batch of data, and the testing accuracy (Figure 1.2), the change in testing accuracy with each batch of data.

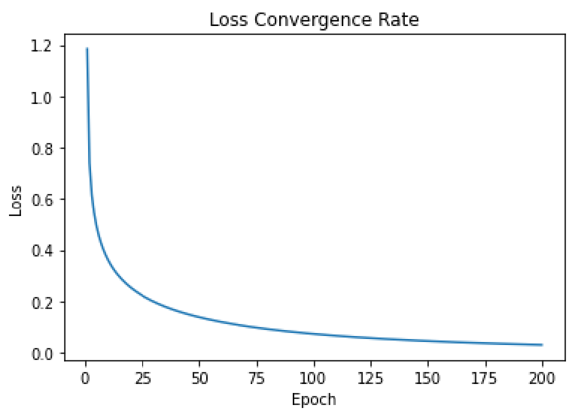

Figure 1.1

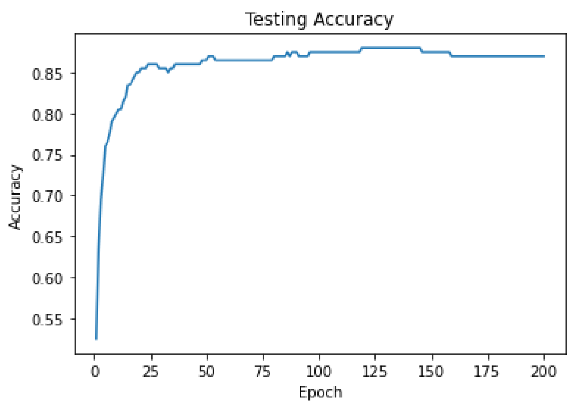

Figure 1.2

Main Challenges

One of the main challenges that we faced in our task 4 were dimension mismatches in matrix multiplication that came from us using the incorrect output matrix as an input matrix for the forward function of a layer. However, we also faced this problem in our backward function for task 4. This was because we did not store the value for the final linear layer from our forward function. This caused us to use the wrong linear layer when we performed backwards on our network. These improper forward and backward calculations also resulted in incorrect loss values, as they did not decrease with each batch in the task 4 checkpoint. Rather, the loss values would be scattered with no general pattern.

We realized that these challenges were also due to an improper processing of our layers_arch parameter in the Network class. We initially separated the linear and nonlinear layers in two separate arrays, but this implementation had become overly complicated and was causing the issues stated above. Hence, to correct this, we stored all of the layers noted in layers_arch in a single array and then solely looped over this array and called the forward and backward functions in the methods for the Network class. This way we were able to update and propagate our outputs to the next layer more concisely, accurately, and inefficiently.

Team Member Contributions

- What we did together:
  - Tasks 4-5
  - Appetizer
  - Debugging All Tasks and Reviewing Report
- Ryan Yong
  - Methodology
  - Network Efficacy
  - Main Challenges
- Vaani Gupta
  - Tasks 1-3
  - Code Implementation
  - Experiment Setup