CS70 Final Project

Technical Report

Harry Beesley-Gilman

Code Implementation and Set-Up.

I can begin by running through my code in each section of the assignment:

I began by filling out methods in the LinearLayer class. The forward method takes an X value, storing it for later backpass and returning X multiplied by the Weights. The backward method computes matrix multiplication for the stored X value and the Y gradient, storing this as W_grad. It then returns the product of the Y gradient and the stored value of W. I use transposes to make the matrices fit together correctly.

For the Nonlinear activation task, I began by filling out the ReLU class methods for nonlinear layers in which we use a ReLU function. In a roundabout way, the forward method copies a matrix, replacing all negative values with 0. The backward method does the same, except with the Y-gradients matrix. Both return the resulting matrix.

For the Loss Function task, I began by filling in the methods of the MSELoss function. The forward method takes a prediction as well as groundtruth, calculating the difference between prediction and groundtruth at each point and summing the squares of these differences in order to return a total value for loss. The backward method

returns the gradient matrix of the loss, which has the same dimensions as the stored data.

Next I built the actual network architecture. I start by initializing the layers instance variable, which is a list of layers. The initialization of the class runs through layers_arch, adding any layers titled “ReLU” as ReLU and all other layers with their actual LinearLayer object. The forward method runs through all layers, propagating input data along through a for loop, finally returning a value for x at the end. Backward starts by taking a reversed list of layers (taking notice of whether we flip the list every time the function is recursively called. It runs backwards propagating the gradient of the output, returning the value at the end.

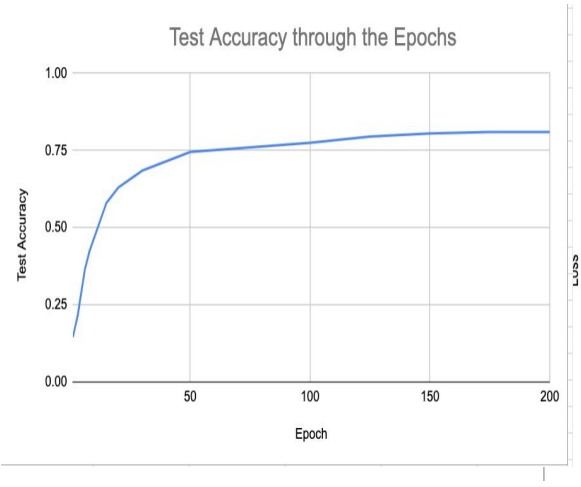
Finally, I moved onto the classification network step. One_Hot_Encode simply turns a list of values into a one-hot zeros matrix 10-wide, filling in a 1 in the appropriate spot to label things. Finally, I adapted class sample code to fill in the train_one_epoc step, merely swapping in the batch of one-hot labels where it is needed in the loss function’s forward step method.

The training of this neural network was completed through 200 Epoch’s; epochs can be thought of as rounds of training. Each layer of the network was either a more standard linear network or a ReLU. I ran my network with the standard batch size (32), classes (10), and epoch number. This resulted in test accuracy values of **81%**. Tweaking any of these parameters would result in performance differences. Here are some examples of the differences parameter tweaking can make:

- A learning rate of 0.05 (5x original value) resulted in much better performance. The model reached **86%** accuracy by the 200th test.
- Doubling the batch size resulted in a minor performance decrease.
- Fewer epochs lead to performance decreases, although returns are diminishing with each additional epoch
- More classes would prove challenging for the neural network and could lead to decreased accuracy in many scenarios.

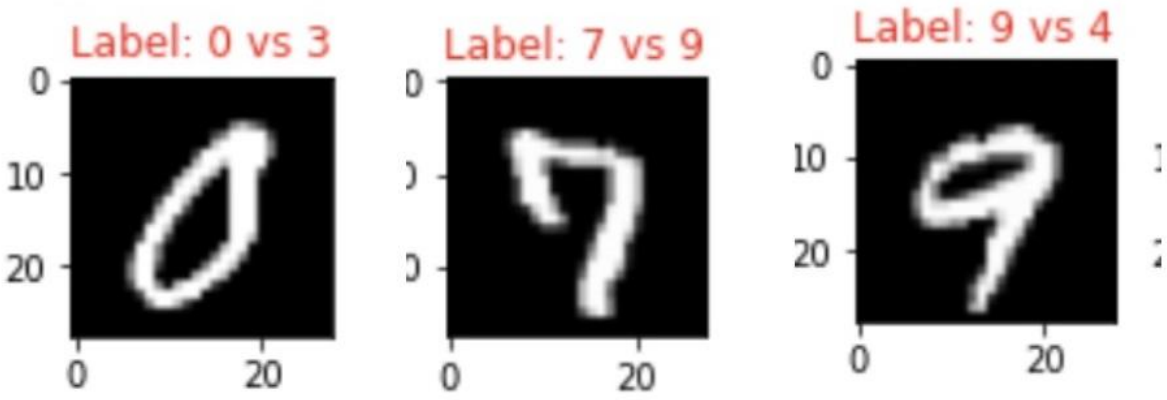
Performance:

Data makes the diminishing returns of each round clear; what starts as enormous improvement with each round turns into less and less notable changes.



Failures:

Some examples of cases in which my neural network failed are shown below:



In the first case, it is not clear why the network failed; perhaps many threes had been written in approximately that orientation within the training data. The second and third examples are a bit more forgivable. The top left of that 7 is shaped much like a nine, and were a few more pixels painted to bridge each side, it would unquestionably be 9. In the third example, 9 vs. 4, I believe the error stems from the one or two pixel gap on top of the nine. Many fours have a similar gap across what would be the top of the number. The beginnings of such a gap could easily trick a machine. All in all, most of the eros looked at least partially similar to the number my network mistook that particular example for.