**Project 1 Report**

**CSC 7700**

**Grant Muslow**

**Abstract**

To implement my MLP, I generally followed the file and went step by step until the whole thing was finished. I ran into significant issues that required hours of debugging, but I ended up with some really good results. For the MPG dataset, I ended with around a 0.05 loss. For the MNIST dataset, I achieved an accuracy of 97.38%

**Methodology**

I approached the problem of coding the MLP head-on. I started with the activation functions, then went on to the layers, and then the full MLP. In addition to all of the methods that were already provided, I created a forward\_with\_layer\_inputs function to the MLP class that returns the layer inputs for each layer. This was used in the backpropagation step where you need layer inputs for each layer. I tested each section thoroughly before moving on to the next, which saved me a lot of headache. However, I ran into significant issues when doing integration testing, which I discuss later. Whenever you instantiate an MLP, I automatically assign the first layer as the input layer, which isn’t affected by backpropagation. During the backpropagation function call, I loop through all layers in reverse order and propagate the deltas and the gradients through the network. Finally, I update the weights and biases.

For the MPG application, I created an MLP with 5 layers, and linear activation at the end. The hidden layers each use ReLU activation and the input layer uses linear activation. I used a dropout of 0.1 in the middle hidden layer. I trained with a learning rate of .001 and 3000 epochs, which took no time at all. I ran into some shaping issues whenever I first tried to run the model, and it led me to add a verbose argument to the MLP so that I could see each step of the training process. This led me to being able to figure out the issue, which was that I forgot to do a transpose when calculating the Hadamard product for non-output layers. After this was fixed, I started getting a training loss of about 0.2, which I thought was okay but I wanted to make it better. I experimented with different dropout values but it didn’t help much. After I printed out the table consisting of true values to predicted, I noticed that my model was only outputting positive predictions, due to my sigmoid activation function at the output. Thus, I changed the activation to linear and started to get really good predictions.

For the MNIST application, I created an MLP with 4 layers, decreasing the fan-out each layer until a softmax layer with 10 outputs. I one-hot encoded the training labels and then ran the model. I got tons of errors. One of which was the shape of the output of my softmax derivative, which was incompatible with my layer backward function. After fixing that, I was still getting crazy losses, so I normalized my data and that fixed the issue. The model was having trouble converging so I increased the learning rate to 0.01 and it converged pretty fast. It took a lot longer per epoch to train the model, which I assume to be because of the data size difference. On my CPU, it took about 3 minutes to do 50 epochs, and I ended up with some really good results.

**Results**

Below are the results for the MPG dataset:

A screen shot of a graph

AI-generated content may be incorrect.

Training loss for epoch 2998 : 0.04775006013219835

Validation loss for epoch 2998: 0.04708764659355692

Starting epoch 2999...

Training loss for epoch 2999 : 0.04430096264989619

Validation loss for epoch 2999: 0.042665307736946816

Total test loss: 0.06387397881527929

Average test loss: 0.06387397881527929

True MPG Predicted MPG

0 -0.329033 -0.028484

1 0.294705 0.906034

2 0.294705 1.095813

3 0.169957 0.628896

4 0.768745 1.550235

5 -0.204286 0.182519

6 -0.466255 -0.266838

7 -0.952771 -0.763398

8 -0.678326 -0.498910

9 -1.077518 -1.517566

For the MNIST Dataset:  
A group of numbers in black squares

AI-generated content may be incorrect.

A graph of a graph

AI-generated content may be incorrect.

Starting epoch 49...

Training loss for epoch 49 : 0.04885913060139625

Validation loss for epoch 49: 0.6135188752044981

Number of correct predictions: 9738

Total number of predictions: 10000

Accuracy: 97.38%

**Code repo link**

<https://github.com/Gmuslow/Foundational-AI>

**Discussion & Conclusion**

This was a very fun project and I learned a lot. The results that I got for both applications were very good to say the least, and this is without any optimizers like RMSProp or ADAM. The dropout helped propel the loss down which led to really good results. I specifically learned a lot about the low-level computations that a neural network does at the batch level, whereas I previously only knew how to do it at the SGD level. I also learned how to take the derivatives of some of these cool activation functions like MISH. All in all, this was a really cool and fun project to learn the basis of all AI applications that exist today.