Linear Perceptron Using the MNIST Data Set

Dalton Rothenberger

1. Data

Data = Normalized, Loss = Binary Cross Entropy, Activation = Softmax, Batch Size = 32, Epochs = 50, Learning Rate = 0.1

Layers	Accuracy	Loss
Single Layer (10)	0.9248	0.0448
2 Layers (Outer = 10, Middle = 5)	0.7553	0.1269
2 Layers (Outer = 10, Middle = 3)	0.5605	0.2003
2 Layers (Outer = 10, Middle = 10)	0.7478	0.1293

Data = Normalized, Loss = MSLE, Activation = ReLU, Batch Size = 32, Epochs = 50, Learning Rate = 0.1

Layers	Accuracy	Loss
Single Layer (10)	0.9176	0.0081
2 Layers (Outer = 10, Middle = 5)	0.7411	0.0163
2 Layers (Outer = 10, Middle = 3)	0.6656	0.0235
2 Layers (Outer = 10, Middle = 10)	0.8576	0.0099

2. Results and Analysis

1. My GitHub account is available here

2. Normalizing the data provided a significant increase in accuracy. Normalizing the data alone increased my accuracy by roughly 80%. The various activation functions produced varying results in term of accuracy. The Sigmoid function and Tanh function produced similar results which makes sense has they are relatively the same graph. The linear activation function produced the worst results of the functions I tried.

3. For a single layer perceptron, Softmax produced the best results. It had the highest accuracy and the lowest loss. In the final form of my single layer perceptron, Softmax produced an accuracy of 91.04% and a loss of 0.0086.

4. The hidden layer architecture that worked best for me was using ReLU with MSLE as the loss function with 10 nodes in the outer layer and 10 nodes in the middle layer. Even though this was the best hidden layer architecture for me it still did not perform better than my single layer version. This was most likely the result of the model being overly complex for the data and not having any drop out introduced.

5. One way of approaching this would be to make multiple multi-class classifiers for each type of label. These classifiers would be separate from one another so the results of one would not influence the other. Based on the example we could have two multi-class classifiers that the input is run through. The first classifier would get the type of animal so ”cat” in this instance. The other classifier would determine the action occurring so in this case ”sleeping”. These two outputs would be independent of each other and could be combined together to get the multi-label classification. A Softmax activation at the prediction layer would be appropriate for this approach. Softmax assigns decimal probabilities to each of the classes in a multi-class problem so a Softmax layer can be put on each of the multi-class classifiers. Softmax requires that each example has exactly one class and since the multiple labels are being broken up into multi-class classifiers that determine one label each this requirement is satisfied.