Linear Perceptron

Dalton Rothenberger

Normalization

Activation = Linear, Loss = MSE, Batch size = 128, Epochs = 20, Learning Rate = 0.1

	Accuracy	Loss
Not Normalized	0.098	nan
Normalized	0.8234	3869.5141

Normalizing the data provided a great increase in accuracy but the loss here can still be improved upon. Normalizing reduced the scale to be between 0-1 without distorting the differences in data or losing an information. This helps helps the gradient descents converge more quickly.

Loss Function

Data = Normalized, Activation = Linear, Batch size = 128, Epochs = 20, Learning Rate = 0.1

	Accuracy	Loss
MSE	0.8234	3869.5141
MSLE	0.5988	2.3324
MAE	0.6811	20.1861
Hinge	0.3486	0.9000
Squared Hinge	0.3157	0.9000

I ended up going with Mean Squared Logarithmic Error because it had the best accuracy while keeping the loss to a minimum. This function is similar to MSE but you take the natural logarithm of the predicted values before calculating the MSE. This relaxes the punishment for large differences in large predicted values. It is best for non-normalized values however it still worked for these normalized values.

Activation

Data = Normalized, Loss = MSLE, Batch size = 128, Epochs = 20, Learning Rate = 0.1

	Accuracy	Loss
Linear	0.5456	1.6985
Softmax	0.8902	0.0105
Sigmoid	0.715	0.0172
Tanh	0.7296	0.0343

Softmax ended up having the best performance. It had the highest accuracy and the lowest loss so I chose this function. The sigmoid function and tanh are roughly the same graph so I am not surprised they performed similarly. From what I found online it looked like softmax was useful in multi-class situations which this scenario is.

Batch Size

Data = Normalized, Loss = MSLE, Activation = Softmax, Epochs = 20, Learning Rate = 0.1

	Accuracy	**Loss**
128	0.881	0.0107
64	0.9003	0.0095
32	0.9048	0.0091
10	0.9086	0.0088

Decreasing the batch size to 10 made the program take much more time and did not improve the performance significantly. I went with a batch size of 32 because it took a reasonable about of time and gave good results.

Epochs

Data = Normalized, Loss = MSLE, Activation = Softmax, Batch Size = 32, Learning Rate = 0.1

	Accuracy	**Loss**
10	0.901	0.0095
20	0.9065	0.0090
30	0.9093	0.0087
50	0.9095	0.0087

Increasing the epochs increased the the overall performance as well. I settled on leaving epochs at 50 since it was it did not run for too long but still gave good results. By increasing the epochs you are increasing the number of times the model is run through the data and each time it performed better.

Learning Rate

Data = Normalized, Loss = MSLE, Activation = Softmax, Batch Size = 32, Epochs 50

	Accuracy	**Loss**
0.01	0.889	0.0106
0.1	0.9104	0.0086
0.5	0.9059	0.0090

The original learning rate was 0.01 but I changed it to 0.1 early on. I believe 0.01 did not make big enough changes to the weights but 0.5 adjusted the weights too much. I found 0.1 to be the best and ended up choosing it for my final model.