The first thing I wanted to alter was the hyperparameter for dropout rate. I didn’t really understand what affect this would have on the performance of this network. I started by setting the dropout rate to 0. This had the immediate effect of making the network learn much slower. Functionally, this makes sense because without any dropouts, every node needs to be analyzed. In order to analyze the data, I changed the way in which the program printed outputs. I commented out the existing print statements, and replaced them with print(“{};{};{}”.format(iteration,val\_lr,\_loss)). This prints the iteration number, the learning rate and the loss with each epoch, separated by semicolons so they can be easily delimited in Excel. The below graphs map the learning rate and loss in arbitrary units vs epoch number for the base case (dropout rate: 0.4).

The case for dropout rate: 0 generated the following graphs.

While the learning rates are approximately equivalent, the loss graphs clearly have a much different start. The loss graph with dropout rate 0.4 has a lot more variability at the start, presumably because the optimization of each neuron doesn’t carry through necessarily due to the dropout. I also ran the neural network with a 0.1 dropout rate and a 0.3 dropout rate, but the loss curves were much different. The accuracy of the network still increased generally, but the loss function never approached the 0.001 value seen for the others (Observed values were 0.18 and 0.09 respectively). A theory as to why this is happening is that the network uses 1024 neurons, and 10% or 30% of these neurons dropping out is not a whole number.

Armed with this knowledge, I decided the 0.4 dropout rate was admirably achieving maximal accuracy while still not taking ages. Therefore, I decided to try out optimizers other than the native Adam optimizer used in the network example we wer e given. I tried three additional optimizers, which had incredibly interesting results: adagrad, gradientdescent and ftrl. The learning rate graphs for all adagrad and gradientdescent were essentially equivalent to those displayed above, but the loss functions looked much different. Below are the graphs of the loss for each of the three different optimizers. They look so much different from the adam optimizer result that I actually reran the adam a few times and got the same results as before.

The ftrl optimizer on the other hand, had a very bizarre reaction to the loss function. The graph is as follows. Along with it is the dropout 0.2 loss graph with the adam optimizer, because it reminded me of what I saw here.

These results are fascinating, and unclear to me in their meaning. Clearly, the optimizer with the lowest loss value is the adam, but the ones with the clearest graphs are the others. This leads me to believe that the demo is relatively well optimized, because the loss function quickly minimizes. If I had the opportunity (i.e. more time) to change things in the future, I’d change network size and structure, as well as run a much longer session to test the other optimizers to see if their loss curves eventually stabilize out.