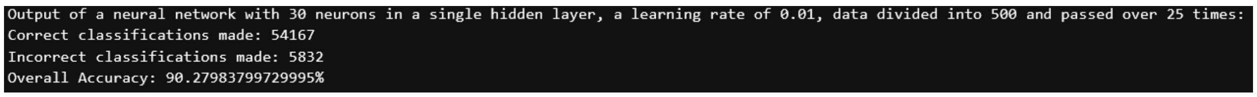
Notes:
-the final setup im going with for .902 accuracy is:
30 neurons in a single hidden layer, .01 learning rate, 25 epochs, 500 batches.

Topology
The Neural Network follows the structure that is requested in the assignment outline; the process begins with 784 input neurons, representing each pixel of a given row's 28x28 image, providing the initial data for the process. This information gets transmitted to the one and only hidden layer of logistic unit neurons, each input providing its weighted value to be summed entered into the activation function of the neuron (logistic), from which the outcome of all neurons will be outputted to the final output layer. Once the hidden layer has submitted its findings to the output layer, we can determine what the system's guess is for the the number displayed by running the output layers softmax function on all the resultant values and seeing which one activates.

Params
-neuron count, learning rate, epochs, batches
My main edited parameters for dialing in accuracy start with the necessities, the count of neurons in the hidden layer as well as the learning rate of each pass, I didn't mess with the initial neuron count I came up with of 30 during my testing period. For learning rate I was able to test the increments of 10, .01 came up as the lead of the other reasonable decimals like .1 and .001, .1 being far too over generalizing and .001 requiring a ludicrous amount of epochs over the data to get me anywhere close to 90% accuracy. Epochs were the number of times the data would be trained over, which easily made this parameter the most important one to tweak around with working on accuracy. The batches parameter determines how many times we pass data from the dataset through the network, the number inputted divides the 10000 samples and takes that many samples for each epoch. Weight initialization is done via a normal distribution created matrix, the size of "input x # of hidden neurons", filled with values running from 0-1 that inputs will be multiplied with the input values when passed to the hidden layer.

Once passed into the hidden layer, forward propagation occurs with the dot product of the inputs and weights being passed into the logistic function of the neuron, dot producting the weights for the next layer (output), then softmaxing the values to keep them in the 0-1.0 range. Back propagation happens next, with the cross entropy of the labels and the outcomes of the hidden layer being calculated and taking the dot product of that result with the weights times the derivative of the logistic function gets us the loss of the trained batch.

Output


```
Output of a neural network with 30 neurons in a single hidden layer, a learning rate of 0.01, data divided into 500 and passed over 25 times:
Correct classifications made: 54167
Incorrect classifications made: 5832
Overall Accuracy: 90.27983799729995%
```

Reflection/Experimentation

I successfully created an artificial neural network that could accomplish the task at hand, but I found there were oddities in my code that I didn't find the time to troubleshoot and optimize. Aside from some simple setup issues throughout my creation process, my information loss per batch completed tends to fluctuate and bounce around the value range of 2-10, without any sort of pattern to it. A previous attempt when my hot encoded labels were all ones by accident proved my system had a functioning info loss system (it stalled at 23 loss in that scenario) but even if I tweak and try to dial in the learning rate and batch sizes this bouncing effect persists. If you run the code I have a statement that prints the loss of each batch's completion to see for yourself.