**Graded Homework 2**

**Code Explanation**

Similar to graded homework 1, I split up the code into different functions to allow me to dynamically generate layers and test parameter values.

The buildSingleLayer function creates a single layer (either ReLU, Sigmoid, or Linear) and applies dropout if specified.

The buildLayers function iterates over all the specified layers and calls buildSingleLayer.

The runAlgorithm function contains the majority of the code that was provided to us.

The main function contains the parameters I selected, and they are placed inside of a Python dictionary. To help with testing, I created a list of these dictionaries to contain the different parameters I wanted to test.

Also to help with testing, I created 3 versions of script 1, and 3 versions of script 2.

1. The final version to turn in with specific values
   1. Files:
      1. Script1.py
      2. Script2.py
2. A version to keep all values constant (not using dropout or regularization) except for trying different number of layers and different number of nodes in each layer. This was used to compare the different network configurations to determine the best for each optimizer.
   1. Files:
      1. Script1\_checkLayerShapes.py
      2. Script2\_AdamcheckLayerShapes.py
3. A version to run in an infinite loop trying different values for each parameter within certain ranges. This version kept track of the current best value for each parameter.
   1. I ran this version after getting a general idea of the best number of layers and number of nodes in each layer from the previous script.
   2. It created 3 output files
      1. a file to keep track of the values for every run
      2. a file to keep track of the best parameters (only updating when it got a better test accuracy than it had previously seen)
      3. a file to keep track of the average % accuracy for a given set of parameters (I ran each set of parameters 3 times to account for variation over multiple runs). I needed to know this because I wanted to select parameters that generally had a high test-accuracy.
   3. Files:
      1. Script1\_automated.py
      2. Script2\_automated.py

Script 3 and Script 4 are not split up into separate functions and only have 1 version for each. I modified the mnist2.py file the professor provided to us that used convolutional neural nets.

**Testing**

Like I mentioned above, I started testing by running the Check Layer Shapes scripts to get a general idea of the number of layers and number of nodes in each layer that worked best. Then I input those parameters into my automated scripts, specified the ranges I wanted to check parameters in, and let it go. I would monitor and every so often stop the script, update the best parameter values, update the ranges, and restart the script.

Below are the best accuracies I achieved:

|  |  |
| --- | --- |
| **Script** | **Best Accuracy** |
| Script1.py | 0.9315 |
| Script2.py | 0.9305 |
| Script3.py | 0.9576 |
| Script4.py | 0.9532 |

I ran into several issues while testing. The first was my learning rate was too high for some configurations, and I was getting stuck in the “0 region” of the ReLU. To troubleshoot, I switched to Sigmoid. After I figured out the problem, I reduced the learning rate for ReLU and that helped provide more accurate results for the checkLayerShape scripts.

The next was that over time the TensorFlow graph became too large and caused my GPU to run out of memory. This was a result of my continually testing different values. Each iteration of my test, TensorFlow would add several more nodes to the graph. To overcome this, I called tf.reset\_default\_graph() in every call to runAlgorithm(). As a result of this, I moved the X and Y tf.placeholder initializations to occur after I reset the graph. This took some research to figure out, but once I added that code the problem went away.

Another issue I ran into occurred when I tested convolutional networks on script3 and script 4. The convolutional neural network caused my GPU to run out of memory. The only solution was to run these scripts on my CPU. To do that I had to specify config parameters when I called tf.InteractiveSession(). I had to specify not to use the GPU with tf.ConfigProto(device\_count={'GPU': 0})