Assignment 3 Report

Goal:

- Starting with a CSV file containing 19,000 entries of data scientists, construct a Neural Network that can predict which data scientists are most likely to try and switch jobs.

Setup/First Steps:

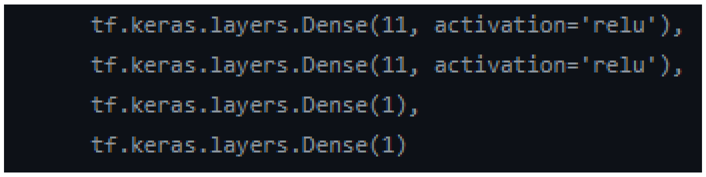
- The first step was to properly install Tensorflow on my machine. This not only required me to install tensorflow, but to install NVIDIA CUDA, a parallel computing platform that allows developers to use video cards for general purpose processing. After discussing the project with other students, it became apparent that If you don't install CUDA, Tensorflow will still run and execute, but it will use the computer's processor instead of it's graphics card.
- This was also my first time ever using Python, so I spent an hour completing a tutorial that was mainly geared towards Java users that aimed at walking through common operations in Java and showing their equivalent operations in Python.
- My plan was first to find a Tensorflow tutorial on Youtube and follow along before attempting to move on to the model required for the assignment. This was not the best course of action, as many of the resourcesI came across were geared more towards data scientists and much of the language they used was above my head. I ended up using the official Tensorflow documentation, which had a few simple tutorials, as a starting point.

Data Cleanup:

- My first attempt at data cleanup involved dropping the first index column and then converting each of the columns with the pandas.Categorical() method. This left me with 11 categories to use in building my model. This was effective at actually allowing my model to compile and start training, but it was not producing desirable accuracy.

- I did some research on one-hot encoding and discovered a method in the pandas library called get_dummies() which automatically applied one-hot encoding to all of the categories in the data file. After using one-hot encoding, i was left with 63 categories

Initial Model:

- Before using one-hot encoding, I had 11 categories. My model would not begin training unless I had at least one layer with 11 neurons. Because of this, my initial model looked something like this: 

```
    tf.keras.layers.Dense(11, activation='relu'),
    tf.keras.layers.Dense(11, activation='relu'),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Dense(1)
```

  . With two hidden layers (11 neurons, and 1 neuron). And an output layer with only one neuron.
- Despite this being my very first model, after running 300 epochs, testing accuracy was still around 75%. Despite attempted improvements, my accuracy would not improve much.

Improvements:

- After implementing one-hot encoding, I experimented with various neuron/layer combinations.
- The instructions in the project statement directed us to have two neurons in the output layer. When i first tried this, I could barely get above 50% accuracy when i was training the model. I discovered that this was because the category assigned as the “target” category was best suited for one neuron if it was kept in it’s initial format. After applying the get_dummies() method to the “target” category, it split the target category into two categories. Once I switched back to two neurons in the output layer, my training and test accuracy started to improve.

Different Iterations:

- As I mentioned before, the first model I constructed without one-hot encoding and rectified linear unit (Relu) as my activation function would leave me with a training accuracy in the mid- 80s (peaked at 86%) if I ran 300 epochs. My data-split for this model was 75/15/15
- My first model after one-hot encoding the data file consisted of:
    - Input layer with 63 neurons with tanh activation.
    - Hidden layer with 63 neurons with tanh activation.
    - Hidden layer with 1 neuron with tanh activation.
    - Output layer with 1 neuron with sigmoid activation.
    - This model would peak at 89-90% training accuracy. I thought that this was peak performance, as I was getting 85% testing accuracy. Upon further examination, I discovered that I was not properly sequestering my training data, so the model had already seen the data that was being tested in the training stage.
- Upon one-hot encoding the target category, I was able to update my output layer to have two neurons, as stated in the instructions. This model consisted of:
    - Input layer with 63 neurons with tanh.
    - Hidden layer with 140 neurons with tanh.
    - Hidden layer with 63 neurons with tanh.
    - Output layer with 2 neurons with sigmoid.
    - This model, and slight variations of it, would serve as the basis for my final model. The requirements of the project stated that we had to have 2 hidden layers. Although, when i removed the second hidden layer, my accuracy in the testing data went from 74 to 76 even though the training accuracy would not

reach as high of a number. There is definitely some risk in overtraining if you try to have too many hidden layers in your network.

Limitations:

- I would be very interested in seeing how my current model would handle a data set of maybe 10x the size. Since the data file is only 19k entries, it is very easy to overtrain the model in a short amount of time. If we were to have 1 million entries. The best way to avoid overtraining would be to run tests every 10/20 epochs and stop the training if the testing accuracy decreased two times in a row.

External Sources:

- For a starting point, i relied heavily on the Tensorflow documentation, particularly, this article: https://www.tensorflow.org/tutorials/load_data/pandas_dataframe
    - Much of the code that i used to read/write to a CSV file was inspired by this link

Future Scope:

- As mentioned above, I would like to apply this model on a much larger data set where I could use different cross validations to make sure that I am not overtraining my model. As of right now, my testing accuracy peaks around 76-77% when I stop it at 100 epochs.