Tyler Walje
Deep Learning Challenge – Module 21

Purpose of Analysis

The purpose of this analysis is to assist the nonprofit foundation Alphabet Soup in selecting optimal applicants to receive funding. Alphabet Soup wants to do their best to select those with the highest chance of success with the funding. In the analysis, we will create multiple neural networks in an attempt to solve this problem.

6 Questions in Results Section

What variable(s) are the target(s) for your model?

The target for the model we're using is "IS_SUCCESSFUL". The "IS_SUCCESSFUL" column simply identifies who is successful or not and is represented as a 1 (successful) or 0 (unsuccessful).

What variable(s) are the features for your model?

The features, after identifying our target, are all other columns (variables) in the dataset. This is because once we know our target, we want to use all of the data available to make the best predictions.

What variable(s) should be removed from the input data because they are neither targets nor features?

The first variables that should be removed from the input data are EIN and NAME. The EIN number is just an identification number that's irrelevant to our predictions. The NAME column is the names of the companies, which are also not necessary in this process.

How many neurons, layers, and activation functions did you select for your neural network model, and why?

For the initial model (deep_learning_final.ipynb/AlphabetSoupCharity.h5) I created a neural network with 80 neurons in the first hidden layer, 30 neurons in the second hidden layer, and an output layer with 1 neuron. We used 'relu' as the activation in the hidden layers and we used the 'sigmoid' activation in the output layer for binary classification. The reason I chose those neuron numbers was that the answers (or preferred output) was already in the starter code. I could see that the starter code output had 80 neurons, 30 neurons, and then 1 neuron. So I simply used that as my starting point.

Were you able to achieve the target model performance?

Unfortunately, I was not able to achieve the target model performance (75%). Even after trying several things to optimize the model, it did not achieve over 75%. I'll talk more about this later when going over the results.

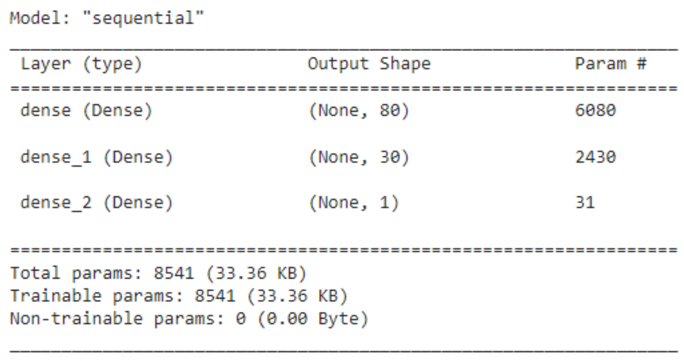
What steps did you take in your attempts to increase model performance?

I did a few things in my attempts to increase model performance. The first thing I did was to increase the number of neurons in the model. I did that because sometimes using more neurons can lead to a more efficient model. The next thing I did was to *decrease* the number of neurons. I wanted to see, after observing the results with more neurons, if fewer neurons would in fact be better. Both led me nowhere. The next thing I did was to use another activation function (LeakyRelu) with the increased number of neurons. The last thing I did was to drop additional columns from what we initially chose to drop. I'll elaborate more on all of this in the next section.

Model-by-Model Results

Let’s go over the results of the different models in order. I will begin with the original model, saved in the repository as deep_learning_final.ipynb/AlphabetSoupCharity.h5.

In this model, the first one I worked on, I began by using the data/output that was generated in the starter code. The starter code already had several things that were “built-in” such as reading in the data and dropping specific columns. It also contained the outputs for the starter code, which displayed things like the structure of the model (as shown below).



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                6080

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 8541 (33.36 KB)
Trainable params: 8541 (33.36 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

This information gave me a great starting point for both the activation function as well as the number of neurons in each layer.

After compiling and running the model there was an accuracy of 72.9%. This was slightly lower than the target accuracy of 75%, so I chose to use a couple different optimization techniques in the next few steps. Let’s move to our first optimization model.

In this model (AlphabetSoupCharity_Optimization.h5), I attempted to optimize the original model by adding more neurons. Instead of 80 neurons, 30 neurons, and 1 neuron in the output layer, I chose to use 150 neurons (first hidden layer), 50 neurons (second hidden layer), and 1 neuron for the output layer. Those numbers were slightly arbitrary, but still based on the original number of neurons. I chose 150 for the first hidden layer because it was a “clean” number that was almost double of the original number of neurons. I chose 50 for the second hidden layer because it seemed like an appropriate number of neurons given the proportion of the original neurons. The 1 hidden layer in the output is standard, so I kept that.

After running 100 epochs, I noticed only a very slight improvement in the model from 72.9% accuracy to 73.0% accuracy. This is such a small increase that we cannot possibly identify this as a “better” model. If anything, it’s performing very similar to the original model, which is not the desired outcome. As a result, I tried something else for the next step.

For my 3rd model (2nd time optimizing) I simply *reduced* the number of neurons in the hidden layers, relative to the original model. I increased the number of neurons in the previous attempt, so I thought it would be interesting, and maybe beneficial, to decrease the number of neurons and see if that helped. I chose to use 50 neurons in the first hidden layer, 15 neurons in the second hidden layer, and 1 neuron for the output layer. Interestingly, the model slightly improved *again*, up to 73.1% accuracy. But the “improved model” was only able to improve by 0.1%. That means that we’ve only achieved a 0.2% increase in accuracy based on our two optimization models so far.

My 4th model (3rd time optimizing), I decided to use the highest number of neurons (150, 50, 1) that I used in my first optimization attempt and a different activation function. Instead of ‘relu’ I chose to use ‘LeakyReLU’ because ‘LeakyReLU’ is a popular function that is known to address the limitation of the ‘relu’ function. Therefore, I thought it would be a great change for this model and would possible increase the accuracy. Once again, unfortunately, the model did not meet the standards we were hoping for, as it only displayed a 72.9% accuracy.

My 5th and final model (4th time optimizing), I chose to drop more columns that we previously dropped. Instead of just dropping the EIN and NAME columns, I also dropped: USE_CASE, ORGANIZATION, and SPECIAL_CONSIDERATIONS. The reason for dropping those columns was to see if the data in those columns was not as necessary or relevant in our prediction process. The USE_CASE column contained the reason for the funding. Some examples are healthcare, preservation, product development etc. The ORGANIZATION column contained the type of organization (trust, association, co-op etc.) and the SPECIAL_CONSIDERATIONS column showed if the company listed had any special considerations. I thought these might be less relevant, but needed to run the model to find out.

In this model, I chose to use all of the original code from the first model but change the dropped columns. I did that because I wanted the cleanest version of the model when seeing what dropping these columns did to the accuracy of the model. The results were disappointing as this model only achieved a 72.8% accuracy. Clearly, this is still short of our goal of 75% accuracy.

Overall Results

Unfortunately, none of the 5 models I created were able to achieve the desired accuracy of 75%. Each time I tried a different optimization technique, the results were so minimally changed (either positively or negatively) that I couldn't reliably say that those techniques were *effective* nor *ineffective.* The accuracy of all the models hovered right around 73% and nothing I did in each optimization attempt really helped or hurt the accuracy and reliability of the model.

Possible Other Model

There are several things that can be done in order to continue to optimize the model. In terms of using the existing notebooks and improving upon them, these things might be impactful:

- Adjust the hidden layers
- Try different activation functions
- Adjusting the number of epochs

Another possible model that could be used for this assignment might be the Random Forest Classifier. The Random Forest Classifier is known to be effective for classification problems and can be more effective in overall model accuracy. If overfitting were to become an issue, the Random Forest Classifier approach is well known to handle overfitting well.