Neural Network Analysis

**Overview**

The purpose of this analysis was to create a tool that would enable Alphabet Soup to select candidates for funding with the best chance of success.

To do this past applicant data was processed consolidating rare cases and outliers of applicants to have useable data where relationships could be established. Categorical data was then turned into individual columns for each unique value and assigned binary values of 1s and 0s. Data from the “Name” and “Ein” columns were removed because they identify an individual candidate, but don’t have any bearing on their success. Once the data was processed the data was split into training and testing data. Our target y-value was based on if an applicant was successful and our x-values (features) were based on all other columns in the dataset. As a last step in preprocessing the data was scaled.

Neural network models were then created, compiled and trained with the data. After each training accuracy and loss of the data was calculated to determine the effectiveness of the neural network in predicting candidate success.

Results:

* Initial Model:
  + 2 hidden layers, first with 80 nodes and the second with 30 nodes. Both used the Relu activation function for the hidden layers. These were the initial inputs provided and were a good starting point for an initial run of the model. The original data set contained 44 initial inputs columns and 34,299 rows. Based on the formula below. Based on the formula below, 80 neurons fits where α is about 10.

Nh=Ns/(α∗(Ni+No))

Ni = number of input neurons.  
No= number of output neurons.  
Ns = number of samples in training data set.  
α = an arbitrary scaling factor usually 2-10.

* + The model used sigmoid for it’s output activation and trained on 100 epochs.
  + The resulting model had a loss of 0.55 and an accuracy of 72.48%
* Optimization Attempt #1
  + This model used the same number of layers, neurons, and activation functions as the first model.
  + In this model, the data for “ASK\_AMT” was categorized into 9 ranges rather than thousands of unique inputs. This was to reduce the number of unique variables in the data’s training.
  + The resulting model had a loss of 0.57 and an accuracy of 72.20%
* Optimization Attempt #2
  + The dataset was reverted back to its original form from the first model.
  + The data used 2 hidden layers the first layer had 380 nodes and the second layer had 200.
  + The hidden layers used the relu function and the output layer used the sigmoid function and ran for 100 epochs.
  + The resulting model had a loss of 0.56 and an accuracy of 72.46%
* Optimization Attempt #3
  + The dataset still used data in the form it was in in the first model.
  + This model had 3 hidden layers, the first had 150 nodes, the second had 75 and the third had 33.
  + The hidden layers used the relu function and the output layer used the sigmoid function.
  + Another change for this model is that it ran for 200 epochs.
  + The resulting model had a loss of 0.56 and an accuracy of 72.39%

Conclusions

Overall the models were not able to reach the target model performance of more than 75%. For future models, it might be worth testing eliminating additional columns of data that do not impact the success of a candidate or muddy the relationships of successful candidates. For example there is a high number of people asking only for $5000. In that pool can be a mix of successful and unsuccessful candidates so this may be an extraneous factor.

The first model was the most successful and upping the number of nodes in each layer did not have a positive impact on the accuracy of the test. Lowering the number of nodes and possibly changing the activation functions could be alternative ways to improve the model.

The data itself may also be imperfect in telling what makes a successful candidate. While there are many inputs, there may be additional factors not captured in the data that explain why a candidate is successful after receiving funding.