Overview

The goal of this project was to use Machine Learning and Neural Networks to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup

Results

- **Data Preprocessing**
    - What variable(s) are the target(s) for your model?

      Since we were predicting if applicants will be successful in the future, our target variable would be whether past applicants were successful (as recorded in the IS_SUCCESSFUL column)

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

      While every column in the dataframe given could be considered a feature, the APPLICATION_TYPE and CLASSIFICATION features were the ones we chose to directly measure the IS_SUCCESSFUL column against

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

      Both EIN and NAME columns were not used as features or targets, since the specific identity of applicants was not relevant for our purposes

- **Compiling, Training, and Evaluating the Model**
    - How many neurons, layers, and activation functions did you select for your neural network model, and why?

      The initial attempt used 3 hidden layers, the first with 80 neurons, the second with 40, and the third with 10. The unique values in APPLICATION_TYPE and CLASSIFICATION were binned, with any having a count below a certain cutoff point in each being aggregated into a single bin labeled “other”, representing “rare” values (cutoff for APPLICATION_TYPE was 528, while the cutoff for CLASSIFICATION was 1883). Each layer used the relu activation function, except for the final “output” layer, which used the Sigmund activation function
      There wasn’t really a reason I chose these other then them “feeling right”. I just wanted a baseline that I could make adjustments to

- Were you able to achieve the target model performance?

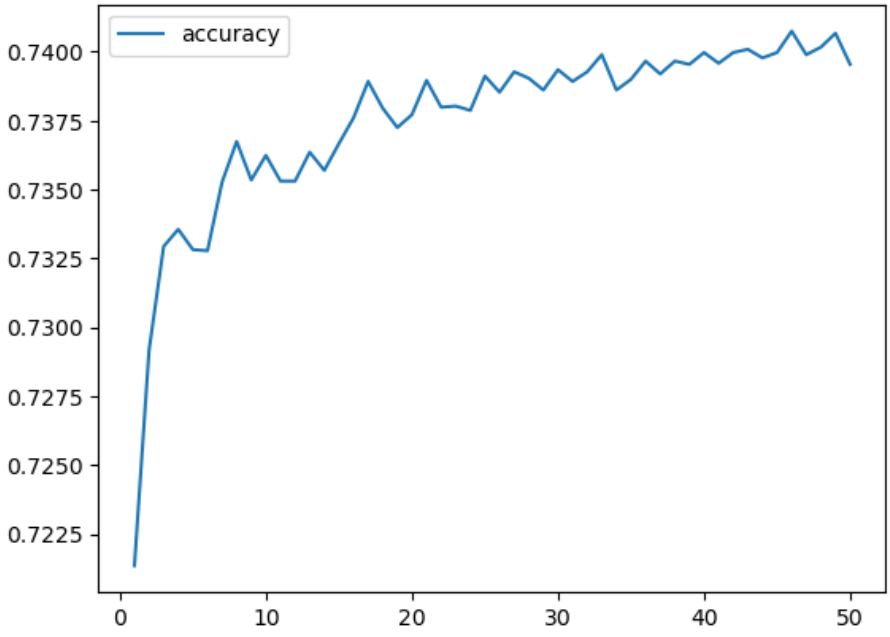
  I was not able to achieve target model performance, getting 72.7% instead (not bad, but not great either)

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

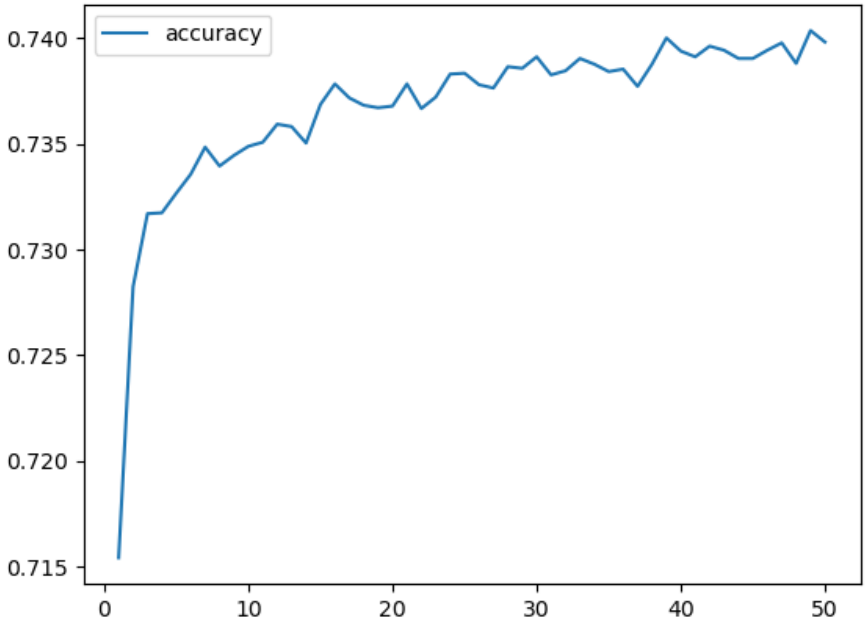
  I tried modifying the number of neurons, number of layers, and the cutoff points for binning the APPLICATION_TYPE and CLASSIFICATION features INDIVIDUALLY to see how each of those factors affected performance.

  (Note that after each of the examples listed below, the settings were reset back to what they were initially set to as seen the first bullet point of this section)

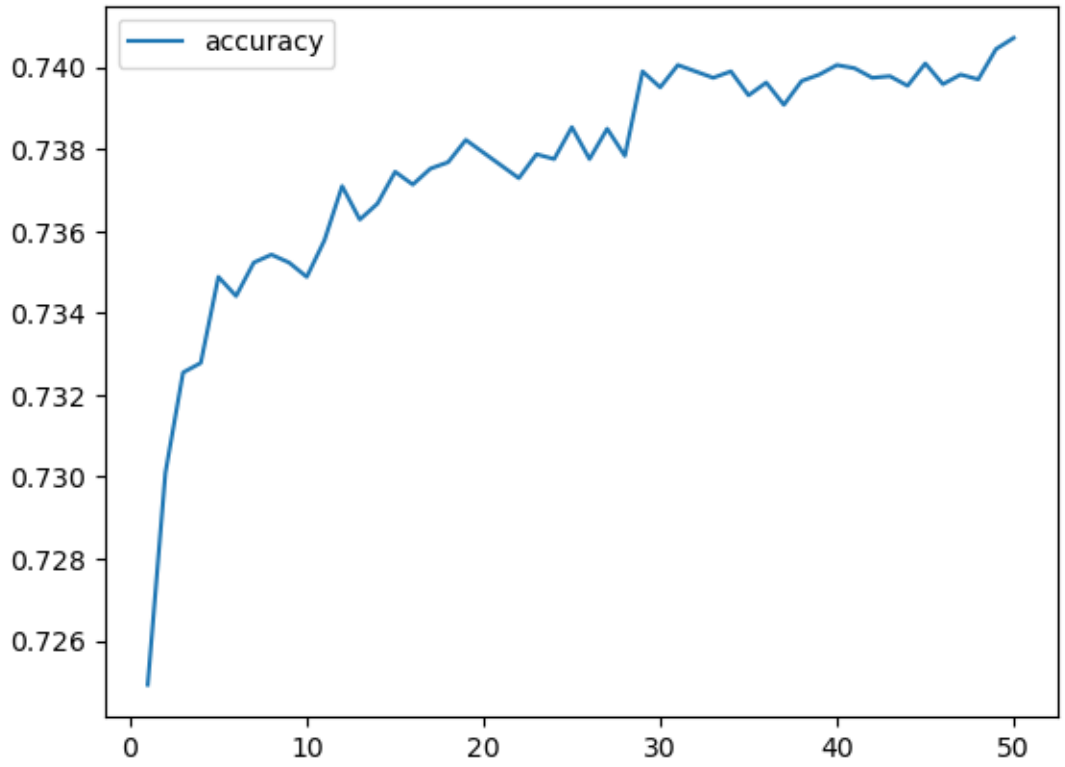
  First, the Cutoff points were changed to 156 for APPLICATION_TYPE and 287 for CLASSIFICATION, resulting in an overall accuracy of 72.6%



Next, I added an extra hidden layer with 5 neurons, which resulted in an overall accuracy of 72.6%



Finally, I gave the first layer 100 neurons , the second layers 64 neurons, and the third layer 16 neurons, resulting in an overall accuracy of 72.8%



One thing to note is that I was quite hesitant to change the activation functions, since I thought that changing it would have a large effect on the overall process and I wanted to experiment with the “simpler” components of the model to start with

Summary

Overall, it was surprisingly hard to change the overall accuracy of the model from around 72.6%.

Based on the above findings, increasing the number of neurons for each layer seemed to have the most positive effect on the accuracy score. In fact, decreasing the cutoff points and increasing the number of layers made the accuracy scores slightly worse. However, I suspect that with the layer experiment specifically, the added layer having 5 less neurons then the preceding one might have something to do with it

Something I want to make clear is that modifying two or more of these factors in conjunction with one another might have more pronounced positive or negative effects on the accuracy score, and it would take quite a bit of time to test each possible combination. I suspect that modifying both the layers and the neurons in each layer will have a more profound positive effect