Project Analysis

**Overview**

Alphabet Soup requested a tool to help select applicants for funding with the highest chance of success. Neural Network models can be used as binary classifiers to determine whether applicants will be successful or not. This requires labeled data to train and test the model, which was provided by the nonprofit with over 34,000 organizations that received funding, and the proper model parameters to get the best outcome. There were 3 attempts made to optimize the model to reach the target 75% accuracy, one of those attempts was aimed at using Keras’s Autotuner.

**Results**

Data Processing

In all attempts, the data was processed the same way. I initially dropped two unnecessary columns within the data, EIN and Name. These two aspects of the data collected won’t give a better predictive model. Data including many different categories were simplified into 6 types by changing the rarer categories into one group, Other. This was done for “classifications” and “application types” data. Once complete, categorical data was changed to True/False and broken into new columns with Panda’s get\_dummies function. The data was separated into two categories, features and targets. The target was what we wanted the model to be able to predict, will the project be successful. The rest of the data was included in features. This cleaned data was then split into testing and training data using train\_test\_split available from Sklearn, followed by using StandardScaler to scale the data, keeping the feature and target data separate.

First Model

The first Sequential neural network model we attempted used an input layer, 2 hidden layers, and an output layer. The input and output layers do not change through manual optimizations, with the input layer having the shape of the incoming data, as well as the output layer having 1 node with a “sigmoid” activation function. The two hidden layers included 7/6 nodes with the “relu” activation function, for a total of 342 parameters, all of which are trainable. It was compiled using the “binary\_crossentropy” loss function, commonly used for binary classification tasks, as well as the “adam” optimizer, and trained over 100 epochs. The results are shown below:



This model provides 72.7% accuracy, 55.9% loss, 71.8% precision, and 80% recall. This doesn’t meet the target 75% accuracy, but it does produce a high recall value when tasked with predicting test data.

First Optimization

The second model uses the same input/output layers and activation functions, but has 3 hidden layers with 8 nodes in each. The model has a total of 481 parameters, all of which are trainable. It was compiled using the same loss function, optimizer, and number of epochs. The results are shown below:



This model provides 73% accuracy, 56.0% loss, 72% precision, and 81% recall. This model is barely better than the first, with a 0.4% increase in accuracy.

Second Optimization

The third model uses the same input/output layers, activation functions, still has 3 hidden layers with 8 nodes in each, but was trained for 200 epochs.The results are shown below:



This model provides 72.4% accuracy, 56.0% loss, 70.1% precision, and 83.3% recall. This model is overall worse than both of the previous attempts, except for an increase in recall of the predicted test data.

Third Optimization - Auto Tuner

The fourth model uses the Keras’s AutoTuner. The data was processed identically to the other models, except it was split another time using the train\_test\_split function to allow for untouched testing data to be used for the final prediction test. The tuner was set to allow for “relu”, “tanh”, and “sigmoid” activation functions, up to 6 hidden layers with 10 nodes each, and finished with a sigmoid output layer. It was trained from 3 to 22 epochs during the tuning. The best model was taken from the tuner and was further trained by 100 epochs. The best model identified by the tuner had “relu” activation functions, and 5 hidden layers with 9/9/3/5/7 nodes. The results are shown below:



The model chosen to be the best had 72.5% accuracy, and 56.2% loss. This model’s accuracy is worse than the initial model and the second model, while ever so slightly outperforming the third model.

**Summary**

The neural network models created couldn’t reach the target of 75% accuracy. The models created also had high loss values of around 56%, which indicates our model hasn’t learned prevalent relationships in the data to predict the target data (is the project successful). I wouldn’t recommend using this model as a tool to predict successful funding applications due to the high loss and less than 75% accuracy, as well as the inability to interpret how the neural network is making these binary classifications.

I would recommend trying a Random Forest machine learning algorithm, as it may have higher accuracy scores. It is also easier to use and is generally very flexible. For example, it handles missing data points and it can handle large and complex datasets without taking a big performance hit. It’s also possible to interpret the model’s schema to find out how it was making those decisions. It can show which features the model ranked highly when making decisions, allowing for future development if the model is successful.