Alphabet Soup Analysis

In this analysis, we will look at the history of 34,000 organizations that Alphabet Soup Foundation has assisted over time. With the use of neural networks and optimization techniques, we will create a tool that will identify organizations that are most likely to remain successful in their ventures.

Three attempts were made to reach a goal of 75% or higher accuracy rate. Below are the results of each attempt:

1. .7277 accuracy, .5542 loss
2. .7282 accuracy, .5577 loss
3. .7769 accuracy, .4659 loss

Preparation of Data

The preprocessing stage for the dataset required eliminating non-essential columns of data. On the first and second attempts, both EIN and Name were eliminated, but since both were unsuccessful at reaching the target accuracy, Name was kept on the third attempt and used as a binning tool in place of Application Type. Any organization with 15 or less interactions with Alphabet Soup were grouped together as “Other” in order to consolidate outliers and focus on organizations with more history, making the dataset easier to work with and helping to reach the over desired accuracy rate.

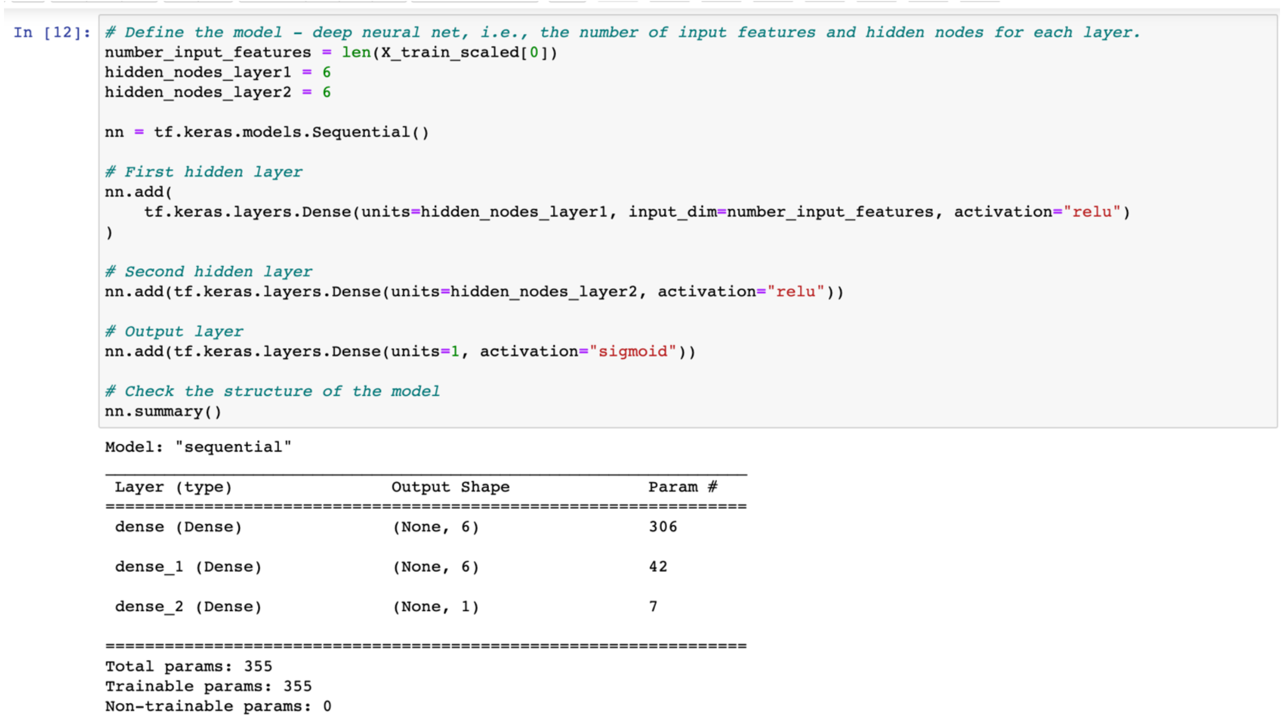
Additionally, in the preprocessing stage, we used the Classification data and raised the binning on the second attempt so that all Classification types less than 1000, were grouped together as “Other”. This did not add value to model results. All categorical data was converted to numerical data. The data was then split. “Is_Successful” was used as our binary classifier, since that is the main question, we are seeking to answer: Will funding a particular organization be successful? All other columns were used as features. Preprocessing was continued by scaling the data, followed by testing, training, and fitting the data to a model.

Building the Model

Using the neural network model, below is how each of the three attempts were set up:

Attempt #1 Inputs

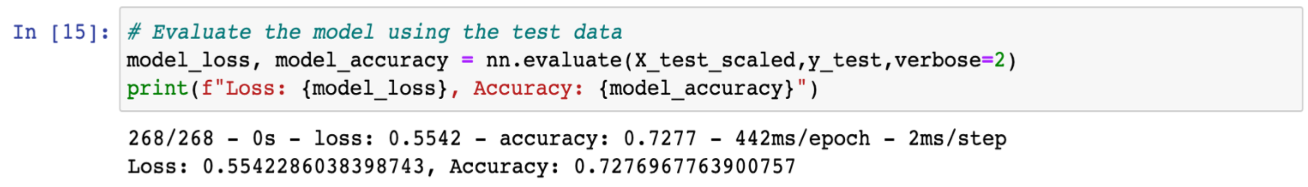
- 2 hidden layers, 6 nodes each
- ReLu activation used on layers 1 and 2, Sigmoid used on the output layer
- 100 Epochs



```
In [12]: # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
         number_input_features = len(X_train_scaled[0])
         hidden_nodes_layer1 = 6
         hidden_nodes_layer2 = 6

         nn = tf.keras.models.Sequential()

         # First hidden layer
         nn.add(
             tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu")
         )

         # Second hidden layer
         nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

         # Output layer
         nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

         # Check the structure of the model
         nn.summary()

         Model: "sequential"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          dense (Dense)               (None, 6)                 306

          dense_1 (Dense)             (None, 6)                 42

          dense_2 (Dense)             (None, 1)                 7

         =================================================================
         Total params: 355
         Trainable params: 355
         Non-trainable params: 0
```

Attempt #1 Results

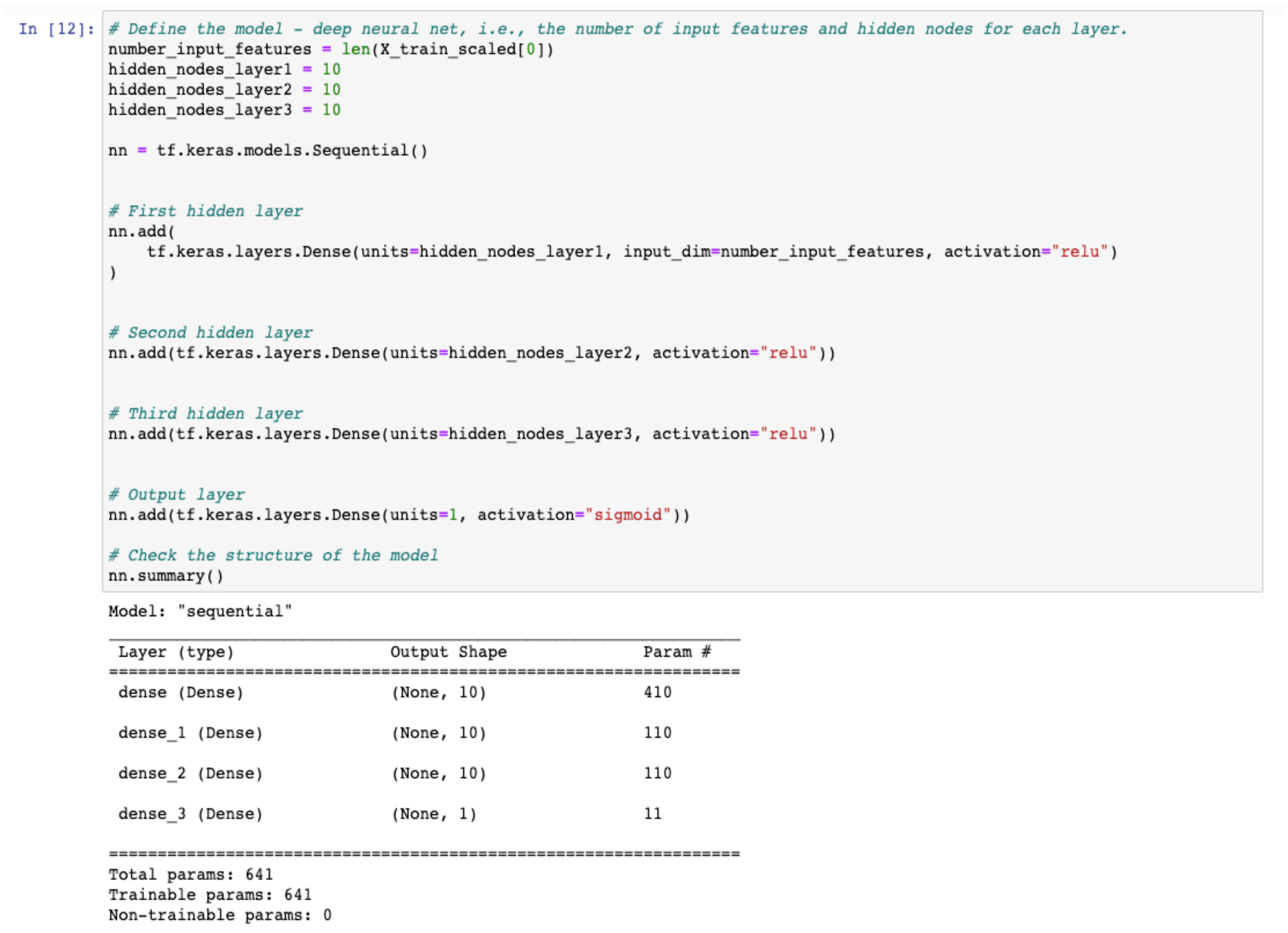
- 72.76% accuracy
- 55.42% loss



```
In [15]: # Evaluate the model using the test data
         model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
         print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

         268/268 - 0s - loss: 0.5542 - accuracy: 0.7277 - 442ms/epoch - 2ms/step
         Loss: 0.5542286038398743, Accuracy: 0.7276967763900757
```

Attempt #2 Inputs

- 3 hidden layers, 10 nodes each
- ReLu activation used on the layers 1, 2 and 3, Sigmoid used on the output layer
- 100 Epochs

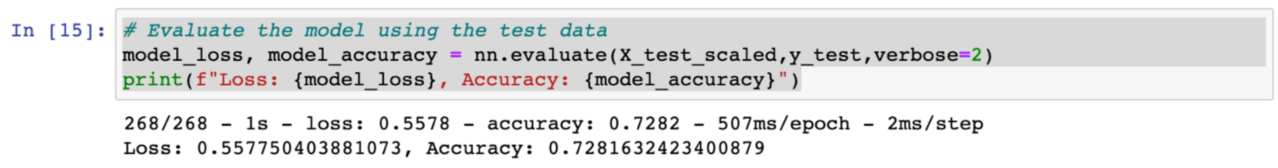


```
In [12]: # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
         number_input_features = len(X_train_scaled[0])
         hidden_nodes_layer1 = 10
         hidden_nodes_layer2 = 10
         hidden_nodes_layer3 = 10

         nn = tf.keras.models.Sequential()


         # First hidden layer
         nn.add(
             tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu")
         )


         # Second hidden layer
         nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))


         # Third hidden layer
         nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation="relu"))


         # Output layer
         nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

         # Check the structure of the model
         nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                410

 dense_1 (Dense)             (None, 10)                110

 dense_2 (Dense)             (None, 10)                110

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
```

Attempt #2 Results

- 72.81% accuracy
- 55.77% loss

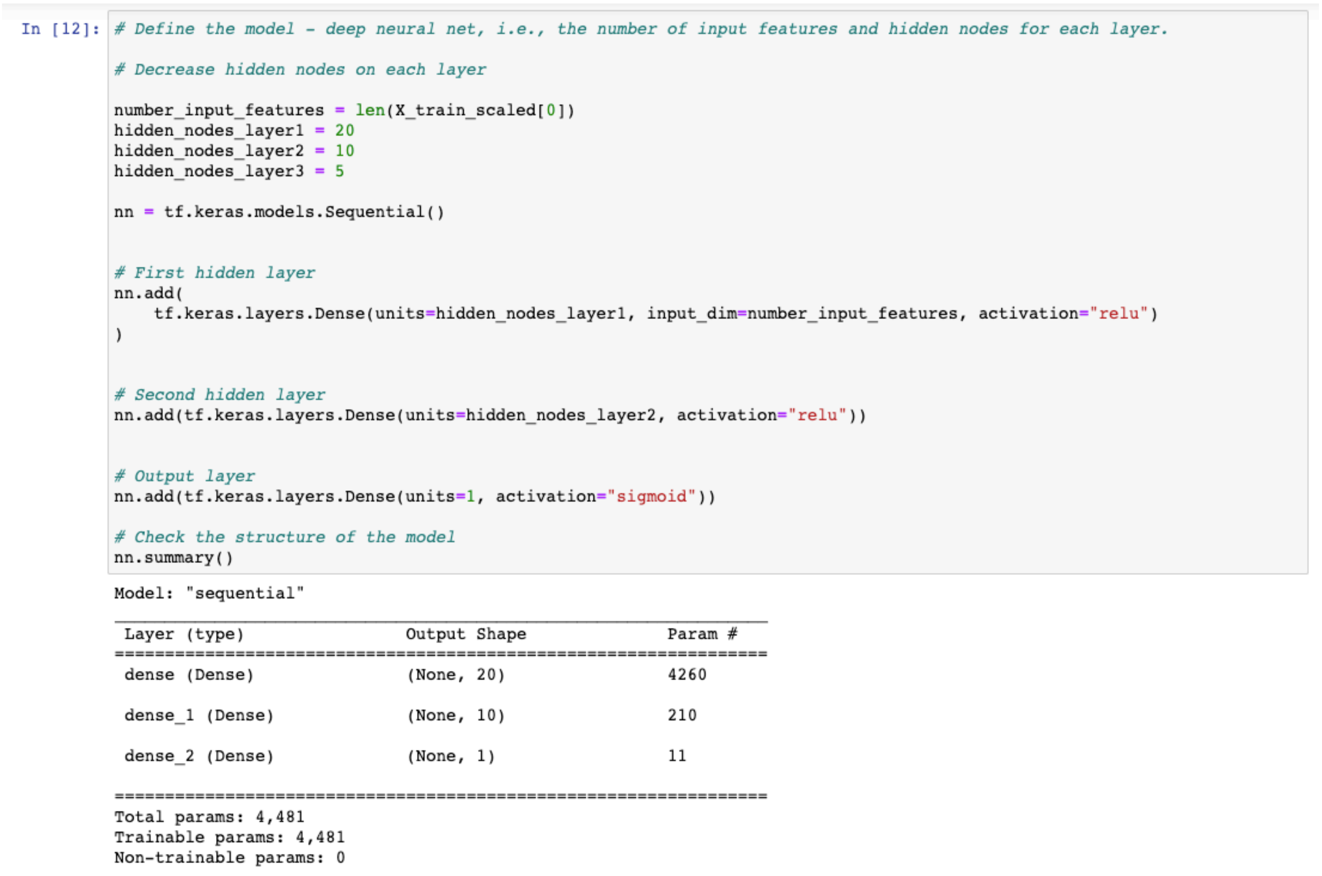


```
In [15]: # Evaluate the model using the test data
         model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
         print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.5578 - accuracy: 0.7282 - 507ms/epoch - 2ms/step
Loss: 0.557750403881073, Accuracy: 0.7281632423400879
```

Attempt #3 Inputs

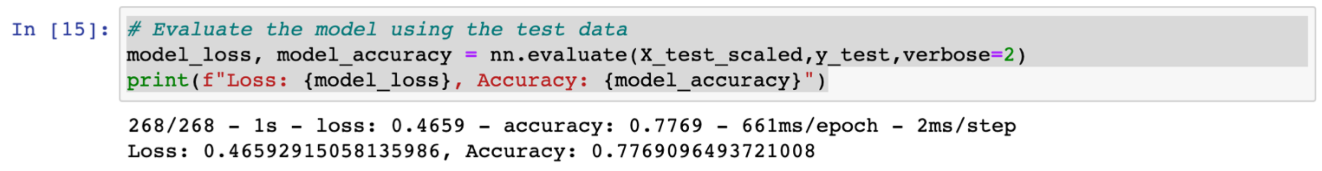
- 2 hidden layers, 20 and 10 nodes
- ReLu activation used on layers 1 and 2, Sigmoid used on the output layer
- 100 Epochs



```
In [12]: # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.

         # Decrease hidden nodes on each layer

         number_input_features = len(X_train_scaled[0])
         hidden_nodes_layer1 = 20
         hidden_nodes_layer2 = 10
         hidden_nodes_layer3 = 5

         nn = tf.keras.models.Sequential()


         # First hidden layer
         nn.add(
             tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu")
         )


         # Second hidden layer
         nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))


         # Output layer
         nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

         # Check the structure of the model
         nn.summary()

         Model: "sequential"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          dense (Dense)               (None, 20)                4260

          dense_1 (Dense)             (None, 10)                210

          dense_2 (Dense)             (None, 1)                 11

         =================================================================
         Total params: 4,481
         Trainable params: 4,481
         Non-trainable params: 0
```

Attempt #3 Results

- 77.69% accuracy
- 46.59% loss



```
In [15]: # Evaluate the model using the test data
         model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
         print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

         268/268 - 1s - loss: 0.4659 - accuracy: 0.7769 - 661ms/epoch - 2ms/step
         Loss: 0.46592915058135986, Accuracy: 0.7769096493721008
```

Summary

While we were able to eventually achieve the desired results of 75% accuracy, we needed to alter much of the data and model in order to do so utilizing drops, binning, additional hidden layers and increasing node count. Using a model better suited for binary classification such as Support Vector Machine or Random Forest Classifier might be a better option at obtaining desired results with less modification to the data and model.