DEEP LEARNING CHALLENGE: CHARITY FUNDING

Project Overview

The nonprofit foundation Alphabet Soup wants a tool that helps with selecting funding applications with the best chance of success in their ventures. Machine learning and neural networks are used to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

Data Preprocessing

The clean up the data, data points that were outliers in columns with too many unique values were binned. For example, the target for the model is the "IS-SUCCESSFUL" column. It signifies if the funding was used effectively. Value of 1 signifies “yes”, while value of zero signifies “no”. Data was split into training and testing sets and categorical variables were encoded using the “get_dummies()” function after successful binning.

```
[11] # Split our preprocessed data into our features and target arrays
     X = application_with_dummies_df.drop(["IS_SUCCESSFUL"], axis='columns').values
     y = application_with_dummies_df["IS_SUCCESSFUL"].values

     # Split the preprocessed data into a training and testing dataset
     X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=78)
```

```
[12] # Create a StandardScaler instances
     scaler = StandardScaler()

     # Fit the StandardScaler
     X_scaler = scaler.fit(X_train)

     # Scale the data
     X_train_scaled = X_scaler.transform(X_train)
     X_test_scaled = X_scaler.transform(X_test)
```

The features of this model are: Name, Application, Type, Affiliation, Classification, Use_Case, Organization, Income_Amt, Special_Considerations, Status, and Ask_Amt. Variable EIN was dropped because the numbers could confuse the model into thinking its significant. SPECIAL_CONSIDERATIONS can be dropped because there is only a small percentage of cases that had any special consideration, and the variable cannot be quantified.

Compiling, Training, and Evaluating the Model

For the optimized model, there are three hidden layers each with many neurons as shown in the image below. The first activation function was 'relu' and the 2nd and 3rd were 'sigmoid'. The output function was 'sigmoid'. Accuracy improved above 72.5% (sequential model accuracy) because the 2nd and 3rd activation functions were 'sigmoid'. Binning outliers in the “NAME” column improved the efficiency. The optimized model has an accuracy of 78.95%.

```
[18] # Define the model - deep neural net
     number_input_features = len(X_train[0])
     hidden_nodes_layer1 =  100
     hidden_nodes_layer2 = 30
     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="sigmoid"))

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

     # 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, 100)               39900

 dense_1 (Dense)             (None, 30)                3030

 dense_2 (Dense)             (None, 10)                310

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 43251 (168.95 KB)
Trainable params: 43251 (168.95 KB)
Non-trainable params: 0 (0.00 Byte)
```

Summary

By increasing the accuracy above 75%, an applicant has about 80% chance of being successful based on the binning requirements used:

- The NAME of the applicant appears more than 5 times.

```
[8]  # Look at NAME value counts for binning
     name_counts = charity_df['NAME'].value_counts()
     #  How many name counts are greater than 5?
     name_counts[name_counts>5]
```

```
PARENT BOOSTER USA INC                                1260
TOPS CLUB INC                                          765
UNITED STATES BOWLING CONGRESS INC                     700
WASHINGTON STATE UNIVERSITY                            492
AMATEUR ATHLETIC UNION OF THE UNITED STATES INC        408
                                                      ...
OLD OAK CLIFF CONSERVATION LEAGUE INC                    6
AMERICAN NEPHROLOGY NURSES ASSOCIATION                   6
HUMBLE ISD EDUCATIONAL SUPPORT GROUPS INC                6
PROFESSIONAL LOADMASTER ASSOCIATION                      6
CBMC INC                                                 6
Name: NAME, Length: 354, dtype: int64
```

- The type of APPLICATION is one of the following; T3, T4, T5, T6, T7, T8, T10, and T19.

```
[11] application_types_to_replace = list(application_counts[application_counts < 500].index)

     # Replace in dataframe
     for app in application_types_to_replace:
         charity_df['APPLICATION_TYPE'] = charity_df['APPLICATION_TYPE'].replace(app,"Other")

     # Check to make sure binning was successful
     charity_df['APPLICATION_TYPE'].value_counts()
```

```
T3       27037
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: APPLICATION_TYPE, dtype: int64
```

- The application has the following CLASSIFICATION; C1000, C2000, C3000, C1200, and C2100.

```
[13] # Determine which values to replace if counts are less than 1000
     classes_to_replace = list(class_counts[class_counts < 1000].index)

     # Replace in dataframe
     for cls in classes_to_replace:
         charity_df['CLASSIFICATION'] = charity_df['CLASSIFICATION'].replace(cls,"Other")

     # Check to make sure binning was successful
     charity_df['CLASSIFICATION'].value_counts()
```

```
C1000    17326
C2000     6074
C1200     4837
Other     2261
C3000     1918
C2100     1883
Name: CLASSIFICATION, dtype: int64
```

Another model that can be used is the Random Forest model. This model is good for classification problems. Using this model produces a 77.6% accuracy.

```
0s
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
```

```
11s [24] # Create a random forest classifier.
     rf_model = RandomForestClassifier(n_estimators=128, random_state=78)

     # Fitting the model
     rf_model = rf_model.fit(X_train_scaled, y_train)

     # Evaluate the model
     y_pred = rf_model.predict(X_test_scaled)
     print(f" Random forest model accuracy: {accuracy_score(y_test,y_pred):.3f}")
```

```
 Random forest model accuracy: 0.776
```