REPORT

BACKGROUND

The purpose of the analysis is to create a training model to provide an accurate prediction of whether prospective funding by Alphabet Soup will yield success or not based on the various existing features. Several variables with numerical and non-numerical values are being considered for the training model.

RESULTS

The data frame column “IS_SUCCESSFUL” was used as the target variable as it is the only true measure of the potential outcome. On the other hand, every column excluding the EIN and NAME columns was included in the features for the model.

EIN and NAME columns had to be dropped because they were deemed neither target nor irrelevant for the features for training the model.

To train the model, initially, 2 hidden layers and one outer layer were used as the optimum starting attributes to use but the overall accuracy ended up not meeting the threshold of 75%. After several tries, adding more hidden layers and playing with more or fewer neurons, the overall accuracy seemed to stick around 72% to 73% - Never even crossing 74%.

BEFORE OPTIMIZATION

These two illustrations show the initial number of hidden layers, the corresponding neurons used and the evaluation results

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each la
number_input_features = len(X_train_scaled[0])
hidden_nodes_layer1 =  80
hidden_nodes_layer2 = 30
outer_layer = 1

nn = tf.keras.models.Sequential()

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

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

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

# Check the structure of the model
```

```
804/804 [==============================] - 1s 1ms/step - loss: 0.5349 - accuracy: 0.7395
Epoch 100/100
804/804 [==============================] - 1s 1ms/step - loss: 0.5352 - accuracy: 0.7394
```

```
[16] # 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: 0.5576 - accuracy: 0.7294 - 343ms/epoch - 1ms/step
Loss: 0.5576224327087402, Accuracy: 0.7294460535049438
```

```
[17] # Export our model to HDF5 file
     nn.save('model_save/AlphabetSoupCharity.h5')
```

AFTER OPTIMIZATION

These Illustrations show the outcome after the next few optimizations. It can be seen that the accuracy isn't affected positively even after adding layers and playing with the neurons.

FIRST OPTIMIZATION

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each la
number_input_features = len(X_train_scaled[0])
hidden_nodes_layer1 =  10
hidden_nodes_layer2 = 20
outer_layer = 1

nn = tf.keras.models.Sequential()

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

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

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


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

```
804/804 [==============================] - 2s 3ms/step - loss: 0.5355 - accuracy: 0.7394
Epoch 100/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5351 - accuracy: 0.7397
```

```
[ ]  # 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: 0.5624 - accuracy: 0.7290 - 543ms/epoch - 2ms/step
Loss: 0.562411904335022, Accuracy: 0.7289795875549316
```

```
[ ]  # Export our model to HDF5 file
     nn.save('model_save/AlphabetSoupCharity.h5')
```

SECOND OPTIMIZATION

```
[14] # Define the model - deep neural net, i.e., the number of input features and hidden node
     number_input_features = len(X_train_scaled[0])
     hidden_nodes_layer1 =  10
     hidden_nodes_layer2 = 20
     hidden_nodes_layer3 = 30
     outer_layer = 1

     nn = tf.keras.models.Sequential()

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

     # 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=outer_layer, activation="sigmoid"))
```

```
804/804 [==============================] - 2s 3ms/step - loss: 0.5344 - accuracy: 0.7400
Epoch 97/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5341 - accuracy: 0.7401
Epoch 98/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5343 - accuracy: 0.7397
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5340 - accuracy: 0.7406
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5341 - accuracy: 0.7401
```

```
# # 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: 0.5539 - accuracy: 0.7298 - 367ms/epoch - 1ms/step
Loss: 0.5538521409034729, Accuracy: 0.7297959327697754
```

```
[18] # Export our model to HDF5 file
     nn.save('model_save/AlphabetSoupCharity_Optimization.h5')
```

THIRD OPTIMIZATION

I then took drastic measures to add more features to the training data set. I decided to add the “NAME” column because there are campaigns with duplicate fundings which would in turn have varied values for the “IS_SUCCESSFUL” column. I then identified the threshold value or cutoff for binning. I grouped all single-funded campaigns into a new name “other” as follows;

```
[4]  application_df["NAME"]

     names_to_replace = []
     all_names = application_df["NAME"].unique().tolist()

     cut_off_3 = 2
     for li in all_names:
       if application_df["NAME"].value_counts()[li] < cut_off_3:
         names_to_replace.append(li)

     # Replace in dataframe
     for ns in names_to_replace:
         application_df["NAME"] = application_df["NAME"].replace(ns,"Other")

     # Check to make sure binning was successful
     application_df["NAME"].value_counts()
```

```
Other                                                                18776
PARENT BOOSTER USA INC                                                1260
TOPS CLUB INC                                                          765
UNITED STATES BOWLING CONGRESS INC                                     700
WASHINGTON STATE UNIVERSITY                                            492
                                                                      ...
WASHINGTON EXPLORER SEARCH AND RESCUE                                    2
INTERNATIONAL ALLIANCE THEATRICAL STAGE EMPLOYEES & MOVING PICTURE       2
BIRTH NETWORK                                                            2
NATIONAL ASSOCIATON OF CORPORATE DIRECTORS                               2
NATIONAL SOCIETY COLONIAL DAMES XVII CENTURY                             2
Name: NAME, Length: 793, dtype: int64
```

After adding the name features to the training dataset, the accuracy dramatically increased to 80% even without a third layer and adding more neurons.

```
# 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 = 20
# hidden_nodes_layer3 = 30
outer_layer = 1

nn = tf.keras.models.Sequential()

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

# 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=outer_layer, activation="sigmoid"))


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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                8450

 dense_1 (Dense)             (None, 20)                220

 dense_2 (Dense)             (None, 1)                 21

=================================================================
Total params: 8,691
```

```
Epoch 98/100
804/804 [==============================] - 2s 2ms/step - loss: 0.3855 - accuracy: 0.8190
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.3856 - accuracy: 0.8179
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.3851 - accuracy: 0.8179
```

```
[17] # # 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: 0.4809 - accuracy: 0.7993 - 693ms/epoch - 3ms/step
Loss: 0.48085442185401917, Accuracy: 0.7993003129959106
```

```
[18] # Export our model to HDF5 file
     nn.save('model_save/AlphabetSoupCharity_Optimization.h5')
```

SUMMARY

I believe more features, relevant or not should be available in the dataset because the more feature rich a data set is, the better the model will learn and provide more accurate outcomes.

At times no matter the number of hidden layers, neurons or the activation function, it is always hard to get a well-trained model unless more features are added. After adding the name column to the training dataset, up to 80% was achieved even after removing an extra neuron that was added.