Charity Funding Report

1. Overview

The non-profit foundation Alphabet Soup wants to select applicants for its charity funding program, and needs to find a system that will help it to select the applicants for funding with the best chance of success in their ventures. The main goal of this analysis is to create a binary classifier by using the appropriate machine learning model that can predict whether or not applicants will be successful if funded by Alphabet Soup. From Alphabet Soup's business team, we received a CSV containing more than 34,000 organizations that have received funding from Alphabet Soup over the years. The dataset contains 12 columns that capture metadata about each beneficiary.

2. Results: Data Preprocessing, and Compiling, Training and Evaluating the Model

Data pre-processing:

- A target is **IS_SUCCESSFUL** for the model to find if the applicants will be successful, if funded.
- For the first attempt, EIN and NAME were dropped and weren't considered being useful. All the features were used except EIN and NAME.
- For optimizing model, EIN, NAME, USE_CASE and SPECIAL_CONSIDERATIONS columns were dropped and all features were used except the columns removed.

Compiling, Training, and Evaluating the Model:

In the first model (Start_code_Ilia.ipynb), I used two hidden layers with 80 and 30 neurons, and a 'relu' activation function for both hidden layers.

```
In [13]: # 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[0])
hidden_nodes_layer1 =  80
hidden_nodes_layer2 = 30
hidden_nodes_layer3 = 1

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, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 5,981
Trainable params: 5,981
Non-trainable params: 0
_________________________________________________________________
```

```
In [14]: # Compile the model
         nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=["accuracy"])
```

```
In [15]: # Train the model
         fit_model = nn.fit(X_train_scaled,y_train,epochs=100)
```

```
858/858 [==============================] - 2s 2ms/step - loss: 0.5365 - accuracy: 0.7410
Epoch 92/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5359 - accuracy: 0.7410
Epoch 93/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5362 - accuracy: 0.7414
Epoch 94/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5360 - accuracy: 0.7405
Epoch 95/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5360 - accuracy: 0.7415
Epoch 96/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5357 - accuracy: 0.7411
Epoch 97/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5358 - accuracy: 0.7403
Epoch 98/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5359 - accuracy: 0.7420
Epoch 99/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5359 - accuracy: 0.7407
Epoch 100/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5357 - accuracy: 0.7411
```

```
In [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}")
```

```
215/215 - 1s - loss: 0.5556 - accuracy: 0.7227 - 604ms/epoch - 3ms/step
Loss: 0.5556166768074036, Accuracy: 0.7227405309677124
```

The model achieves an accuracy of 72%, which is below the 85% accuracy required for a basic model. In order to improve the performance of the model, another optimizing model was built (AlphabetSoupCharity_Optimzation.ipynb) starting by dropping four columns of EIN, NAME, USE_CASE and SPECIAL_CONSIDERATIONS and increased the number of neurons in the two hidden layers.

Compile, Train and Evaluate the Model

```
In [13]: # 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[0])
         hidden_nodes_layer1 =  90
         hidden_nodes_layer2 = 50
         hidden_nodes_layer3 = 1

         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, 90)                3330

 dense_1 (Dense)             (None, 50)                4550

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 7,931
Trainable params: 7,931
Non-trainable params: 0
_________________________________________________________________
```

```
In [15]: # Train the model
fit_model = nn.fit(X_train_scaled,y_train,epochs=100)
```

```
858/858 [==============================] - 2s 3ms/step - loss: 0.5406 - accuracy: 0.7373
Epoch 92/100
858/858 [==============================] - 2s 3ms/step - loss: 0.5398 - accuracy: 0.7373
Epoch 93/100
858/858 [==============================] - 2s 3ms/step - loss: 0.5400 - accuracy: 0.7375
Epoch 94/100
858/858 [==============================] - 2s 3ms/step - loss: 0.5404 - accuracy: 0.7371
Epoch 95/100
858/858 [==============================] - 3s 3ms/step - loss: 0.5401 - accuracy: 0.7366
Epoch 96/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5400 - accuracy: 0.7377
Epoch 97/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5408 - accuracy: 0.7378
Epoch 98/100
858/858 [==============================] - 2s 3ms/step - loss: 0.5399 - accuracy: 0.7373
Epoch 99/100
858/858 [==============================] - 2s 2ms/step - loss: 0.5400 - accuracy: 0.7374
Epoch 100/100
858/858 [==============================] - 2s 3ms/step - loss: 0.5399 - accuracy: 0.7369
```

```
In [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}")
```

```
215/215 - 1s - loss: 0.5686 - accuracy: 0.7273 - 889ms/epoch - 4ms/step
Loss: 0.5685807466506958, Accuracy: 0.7272594571113586
```

However, the performance of the model was not improved. The loss slightly increased from 0.5556 to 0.5686, and the accuracy rate increased slightly from 0.7227 to 0.7273.

3. Summary

The model's best accuracy rate was 73%, and after building another model, the value was not improved by increasing the number of layers, or dropping more columns in the dataset.