# Deep Learning Challenge

## Overview

The purpose of the charity funding analysis for Alphabet Soup was to predict whether the funding for a particular charity would be successful or not. The goal was to use machine learning and neural networks to apply target/features on the dataset, create a binary classifier that is capable of predicting whether investors would be successful if funded by Alphabet Soup. We started with 34,000 organizations and 12 columns that captured the metadata about each organization and their past funding outcomes.

## Results

Data Preprocessing

- What variable(s) are the target(s) for your model?
    - **IS_SUCCESSFUL** is the target that is marked 1 for successful and 0 for not successful.
- What variable(s) are the features for your model?

  - APPLICATION_TYPE
  - AFFILIATION
  - CLASSIFICATION
  - USE_CASE
  - ORGANIZATION
  - STATUS
  - INCOME_AMT
  - SPECIAL_CONSIDERATIONS
  - ASK_AMT

- What variable(s) should be removed from the input data because they are neither targets nor features?
    - **EIN** and **Name** should be removed from the input data because they are neither targets nor features

- Compiling, Training, and Evaluating the Model
    - How many neurons, layers, and activation functions did you select for your neural network model, and why?

- **Model 1:** Layer1- nodes=80, activation function=relu; Layer 2- nodes=50, activation function=relu

```
[86] # 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 = 50

     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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 80)                3520

 dense_4 (Dense)             (None, 50)                4050

 dense_5 (Dense)             (None, 1)                 51

=================================================================
Total params: 7621 (29.77 KB)
Trainable params: 7621 (29.77 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
804/804 [==============================] - 1s 2ms/step - loss: 0.5353 - accuracy: 0.7407
Epoch 93/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5355 - accuracy: 0.7406
Epoch 94/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5357 - accuracy: 0.7407
Epoch 95/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5353 - accuracy: 0.7410
Epoch 96/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5353 - accuracy: 0.7418
Epoch 97/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5355 - accuracy: 0.7414
Epoch 98/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5354 - accuracy: 0.7418
Epoch 99/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5354 - accuracy: 0.7416
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5349 - accuracy: 0.7410
```

```
[89] # 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: 0.5597 - accuracy: 0.7262 - 460ms/epoch - 2ms/step
Loss: 0.5596936345100403, Accuracy: 0.7261807322502136
```

- **Model 2:** Layer 2 nodes increased to 70, Layer 1 activation function “tanh” instead of “relu”

```python
# 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 = 70

nn = tf.keras.models.Sequential()

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

# 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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 80)                3520

 dense_7 (Dense)             (None, 70)                5670

 dense_8 (Dense)             (None, 1)                 71

=================================================================
Total params: 9261 (36.18 KB)
Trainable params: 9261 (36.18 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
804/804 [==============================] - 1s 2ms/step - loss: 0.5341 - accuracy: 0.7421
Epoch 95/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5344 - accuracy: 0.7410
Epoch 96/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5344 - accuracy: 0.7419
Epoch 97/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5340 - accuracy: 0.7419
Epoch 98/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5342 - accuracy: 0.7415
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5340 - accuracy: 0.7411
Epoch 100/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5337 - accuracy: 0.7420
```

```python
# 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: 0.5593 - accuracy: 0.7258 - 436ms/epoch - 2ms/step
Loss: 0.5593082904815674, Accuracy: 0.7258309125900269
```

- **Model 3:** Same as model 2 but decreased epochs to 50

```
804/804 [==============================] - 1s 2ms/step - loss: 0.5324 - accuracy: 0.7423
Epoch 41/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5322 - accuracy: 0.7425
Epoch 42/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5318 - accuracy: 0.7421
Epoch 43/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5322 - accuracy: 0.7425
Epoch 44/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5318 - accuracy: 0.7419
Epoch 45/50
804/804 [==============================] - 2s 2ms/step - loss: 0.5321 - accuracy: 0.7424
Epoch 46/50
804/804 [==============================] - 2s 2ms/step - loss: 0.5320 - accuracy: 0.7420
Epoch 47/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5321 - accuracy: 0.7425
Epoch 48/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5316 - accuracy: 0.7426
Epoch 49/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5318 - accuracy: 0.7425
Epoch 50/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5319 - accuracy: 0.7421
```

```python
# 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: 0.5687 - accuracy: 0.7259 - 323ms/epoch - 1ms/step
Loss: 0.5686773618283245, Accuracy: 0.7259474092752075
```

Were you able to achieve the target model performance?

The target for the model achieved a 72% model performance despite optimization of the code.

What steps did you take in your attempts to increase model performance?

In the optimization, I attempted change the number of hidden layers, change activation function and reduce the epochs. The accuracy did not reach 75%.

## Summary

Based on the models created, the accuracy remained unchanged even after changing number of layers, activation functions, the number of epochs. As the random forest classifier is less affected by outliers, it should be the next step.