# Deep Neural Networks

## Overview:

The nonprofit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. With your knowledge of machine learning and neural networks, you'll use the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

## Questions:

## Variables in my model:
APPLICATION_TYPE
CLASSIFICATION

## The targets in my model:

```
[11] # Split our preprocessed data into our features and target arrays
     X = application_df.drop('IS_SUCCESSFUL', axis=1).values
     y = application_df['IS_SUCCESSFUL'].values

     # Split the preprocessed data into a training and testing dataset
     # 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)
```

## The features of my model are as follows:

```
[4]  # Determine the number of unique values in each column.
     for each in application_df.columns:
         print(each, len(application_df[each].unique()))
```

```
APPLICATION_TYPE 17
AFFILIATION 6
CLASSIFICATION 71
USE_CASE 5
ORGANIZATION 4
STATUS 2
INCOME_AMT 9
SPECIAL_CONSIDERATIONS 2
ASK_AMT 8747
IS_SUCCESSFUL 2
```

What variables were removed because they were neither features or targets:
AFFILIATION
ORGANIZATION
STATUS
USE_CASE
INCOME_AMT
SPECIAL_CONSIDERATIONS
ASK_AMT

COMPILING, TRAINING AND EVALUATION THE MODEL

I used the following neurons, layers and activation functions for my neural network model:

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
# Define the model - deep neural net
number_input_features = len(X_train[0])
hidden_nodes_layer1 =  5
hidden_nodes_layer2 = 8
hidden_nodes_layer3 =11

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"

Results: I was not able to achieve the target model performance:

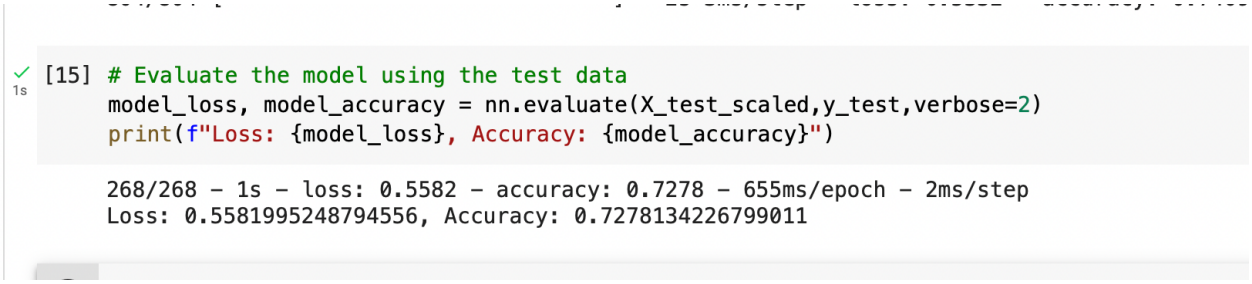
```
[16] # Evaluate the model using the test data
     model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
     print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

     268/268 - 1s - loss: 0.5565 - accuracy: 0.7248 - 733ms/epoch - 3ms/step
     Loss: 0.5564953684806824, Accuracy: 0.724781334400177
```

To optimize to try and get better performance, I did the following:

1. I lessened the amount of epochs
2. Added more neurons
3. Added additional layers

Unfortunately, I still did not receive the target model performance (as shown):



```
[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: 0.5582 - accuracy: 0.7278 - 655ms/epoch - 2ms/step
     Loss: 0.5581995248794556, Accuracy: 0.7278134226799011
```

Summary: I did not receive the target model performance. I may have had better performance had I added even more neurons and layers. Not sure what other models could have been used to enhance performance.