# Deep Learning Challenge- Alphabet Soup

# Neural Network Model

## Overview:

The non-profit 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 knowledge of machine learning and neural networks, we are required to 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.

## Data Processing:

What variable(s) are the target(s) for your model?

In my model, the target variable was labelled "IS_SUCCESSFUL" and has the value of 1 for yes and 0 for no. "APPLICATION_TYPE" data was analysed and "CLASSIFICATION" value was used for binning.

```
In [11]: # Split our preprocessed data into our features and target arrays
y = application_df['IS_SUCCESSFUL'].values
y

# Take out IS_SUCCESSFUL
X = application_df.drop('IS_SUCCESSFUL', axis=1).values
X

# 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 = 42)
```

In [25]:
```python
# Choose a cutoff value and create a list of application types to be replaced
# use the variable name `application_types_to_replace`
application_types_to_replace = list(counts_app[counts_app<500].index)
application_types_to_replace

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

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

Out[25]:
```
T3       27037
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: APPLICATION_TYPE, dtype: int64
```

In [28]:
```python
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`
classifications_to_replace = list (counts_binning[counts_binning<1880].index)
classifications_to_replace


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

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

Out[28]:
```
C1000    17326
C2000     6074
C1200     4837
Other     2261
C3000     1918
C2100     1883
Name: CLASSIFICATION, dtype: int64
```

In [29]:
```python
# Convert categorical data to numeric with `pd.get_dummies`
application_df = pd.get_dummies(application_df,dtype=float)
application_df.head()
```

## What variable(s) are the features for your model? What variable(s) should be removed from the input data because they are neither targets nor features?

In the initial model I removed “EIN” and “Name” columns and the remaining columns were considered to be features for the model. In addition to this, when I tried to optimise the model, I removed an additional column “ORGANIZATION” and the remaining columns were used as features for the model.

In [21]:
```python
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(columns = ['EIN', 'NAME'])
application_df
```

Out[21]:

| | APPLICATION_TYPE | AFFILIATION | CLASSIFICATION | USE_CASE | ORGANIZATION | STATUS | INCOME_AMT | SPECIAL_CONSIDERATIONS | ASK_AMT |
|---|---|---|---|---|---|---|---|---|---|
| 0 | T10 | Independent | C1000 | ProductDev | Association | 1 | 0 | N | 5000 |

In [2]:
```python
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(columns = ['EIN', 'NAME', 'ORGANIZATION'])
application_df
```

Out[2]:

| | APPLICATION_TYPE | AFFILIATION | CLASSIFICATION | USE_CASE | STATUS | INCOME_AMT | SPECIAL_CONSIDERATIONS | ASK_AMT | IS_SUCCESSFUL |
|---|---|---|---|---|---|---|---|---|---|

## Compiling, Training, and Evaluating the Model:

How many neurons, layers, and activation functions did you select for your neural network model, and why? Were you able to achieve the target model performance?

I used 2 hidden layers. Layer one had 80 neurons and layer 2 had 30 neurons. I also selected "relu" activation function for both hidden layers as it doesn't allow for the activation of all of the neurons at the same time therefore easy for computation. Overall, I chose these features in order to aim to get the required accuracy. However, I wasn't successful of achieving the target model performance of 75%.

### Compile, Train and Evaluate the Model

In [32]:
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_layer = len(X_train_scaled[0])
hidden_nodes_L1= 80
hidden_nodes_L2 = 30

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L1, activation="relu", input_dim=input_layer))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L2, 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, 30)                2430

 dense_5 (Dense)             (None, 1)                 31

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

In [35]:
```
# 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: 0.5583 - accuracy: 0.7296 - 306ms/epoch - 1ms/step
Loss: 0.5583381056785583, Accuracy: 0.7295626997947693
```

In [36]:
```
# Export our model to HDF5 file
nn.save("AlphabetSoupCharity.h5")
```

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

**Optimise attempt 1:**

- Reduce number of columns: removed the organisation column
- Reduce number of bins, decreasing the values for each bin: changing the APPLICATION_TYPE threshold from 500 to 1000.
- Adding the neurons to hidden layer: adding 20 additional nodes to both 1st and 2nd hidden layer

```
In [2]: # Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(columns = ['EIN', 'NAME', 'ORGANIZATION'])
application_df
```

```
Out[2]:     APPLICATION_TYPE     AFFILIATION  CLASSIFICATION  USE_CASE  STATUS  INCOME_AMT  SPECIAL_CONSIDERATIONS  ASK_AMT  IS_SUCCESSFUL
```

```
In [6]: # Choose a cutoff value and create a list of application types to be replaced
# use the variable name `application_types_to_replace`
application_types_to_replace = list(counts_app[counts_app<1000].index)
application_types_to_replace

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

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

```
Out[6]: T3       27037
Other     2266
T4        1542
T6        1216
T5        1173
T19       1065
Name: APPLICATION_TYPE, dtype: int64
```

```
In [13]: # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_layer = len(X_train_scaled[0])
hidden_nodes_L1= 100
hidden_nodes_L2 = 50

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L1, activation="relu", input_dim=input_layer))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L2, 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, 100)               3700

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 8,801
Trainable params: 8,801
Non-trainable params: 0
```

```
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}")

268/268 - 0s - loss: 0.5738 - accuracy: 0.7250 - 274ms/epoch - 1ms/step
Loss: 0.5738229155540466, Accuracy: 0.7250145673751831
```

**Optimise attempt 2:**

- Reduce number of columns: removed the organisation column
- Reduce number of bins, decreasing the values for each bin: changing the APPLICATION_TYPE threshold from 500 to 1000.
- Adding the neurons to hidden layer: adding 20 additional nodes to both 1st and 2nd hidden layer
- Add more hidden layers: adding 1 additional hidden layer
- Use different activation functions for the hidden layers: using sigmoid for all the layers

```
In [25]: # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_layer = len(X_train_scaled[0])
hidden_nodes_L1= 100
hidden_nodes_L2 = 50
hidden_nodes_L3 = 10

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L1, activation="sigmoid", input_dim=input_layer))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L2, activation="sigmoid"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L3, activation="sigmoid"))

# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

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

Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 100)               3700

 dense_12 (Dense)            (None, 50)                5050

 dense_13 (Dense)            (None, 10)                510

 dense_14 (Dense)            (None, 1)                 11

=================================================================
Total params: 9,271
Trainable params: 9,271
```

```
In [28]: # 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: 0.5683 - accuracy: 0.7266 - 267ms/epoch - 997us/step
Loss: 0.5682613253593445, Accuracy: 0.7266472578048706

In [29]: # Export our model to HDF5 file
nn.save("AlphabetSoupCharity_Optimisation.h5")
```

**Optimise attempt 3:**

- Reduce number of columns: removed the organisation column
- Reduce number of bins, decreasing the values for each bin: changing the APPLICATION_TYPE threshold from 500 to 1000.
- Adding the neurons to hidden layer: adding 20 additional nodes to both 1st and 2nd hidden layer
- Add more hidden layers: adding 1 additional hidden layer
- Use different activation functions for the hidden layers: using sigmoid for all the layers
- Add or reduce the number of epochs to the training regimen: increasing the number of epochs from 100 to 200

In [21]:
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_layer = len(X_train_scaled[0])
hidden_nodes_L1= 100
hidden_nodes_L2 = 50
hidden_nodes_L3 = 10

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L1, activation="sigmoid", input_dim=input_layer))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L2, activation="sigmoid"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_L3, activation="sigmoid"))

# 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_7 (Dense)             (None, 100)               3700

 dense_8 (Dense)             (None, 50)                5050

 dense_9 (Dense)             (None, 10)                510

 dense_10 (Dense)            (None, 1)                 11

-----------------------------------------------------------------
```

In [23]:
```
# Train the model
fit_model = nn.fit(X_train_scaled, y_train, epochs=200)
```

In [24]:
```
# 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: 0.5771 - accuracy: 0.7254 - 269ms/epoch - 1ms/step
Loss: 0.5771234035491943, Accuracy: 0.7253644466400146
```

## Summary:

The first optimise model gave me an accuracy of 72.50%, the second attempt gave me 72.6% and the third attempt gave me 72.53%. Therefore, the second attempt provided me the most accurate results even though it failed to reach target accuracy of 75% or higher. One recommendation, on how a different model could achieve the target accuracy could be to add back removed columns such as “NAME” and reduce the number of neurons for layers which will result in lower number of parameters. A model used on lower parameters could increase the accuracy as its working with a smaller dataset.