# Deep Learning Project: Charity Funding Predictor

## Overview

The purpose of this analysis was to help nonprofit foundation Alphabet Soup select the applicants for funding with the best chance of success. Using knowledge of machine learning and neural networks, I used the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful.

## Results

### Data Preprocessing

For data processing the columns EIN and NAME were removed. For the targets in the model [IS_SUCCESSFUL] column was used because of the values 0 or 1. APPLYCATION_TYPE and CLASSIFICATION were used for binning. Used pd.get dummies to convert all categorical value to numeric.

For the AlphabetSoupCharity_Optimization.ipynb data pre-processing we only removed the EIN. Ran all other targets, bins, and pd.get dummies the same as stated above.

### Compiling, Training, and Evaluating the Model

Neural network was applied on each model multiple layers two in total. The input (node) feature was calculated from length X_train, for the model 2 hidden layers with 80, 30 neurons split with activation function relu. Output node is 1 as it was binary classifier model with only one output: was the funding application succesfull yes or no. The output layer activation of sigmoid was used since the model output is binary classification between 0 and 1.

## Compile, Train and Evaluate the Model

In [12]:
```
# 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

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()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense (Dense) | (None, 80) | 3,520 |
| dense_1 (Dense) | (None, 30) | 2,430 |
| dense_2 (Dense) | (None, 1) | 31 |

```
Total params: 5,981 (23.36 KB)
Trainable params: 5,981 (23.36 KB)
Non-trainable params: 0 (0.00 B)
```

The two layer training model generated 5,981 parameters. The first attempt came close at 72%, which was under the desired 75%.

In [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 - 3ms/step - accuracy: 0.7235 - loss: 0.5556
Loss: 0.5555936098098755, Accuracy: 0.723498523235321
```

## Optimization

The second attempt added NAME back into the dataset. Also, the first hidden layer changed to 90, and 30 neurons split with activation function tahn. Everything else stayed the same.

### Compile, Train and Evaluate the Model

In [51]:
```
# 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 = 30

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="tanh")
)

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

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

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_7"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_21 (Dense) | (None, 90) | 40,230 |
| dense_22 (Dense) | (None, 30) | 2,730 |
| dense_23 (Dense) | (None, 1) | 31 |

```
Total params: 42,991 (167.93 KB)
Trainable params: 42,991 (167.93 KB)
Non-trainable params: 0 (0.00 B)
```

The two layer training model generated 42,991 parameters. The optimized attempt came at 79%, which was over the desired 75%.

In [54]:
```
# 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 - 3ms/step - accuracy: 0.7906 - loss: 0.4411
Loss: 0.4411293864250183, Accuracy: 0.7905539274215698
```

# Summary

The optimized neural network trained model from the keras turner method achieved close to 79% prediction accuracy with a 0.44 loss, using a sigmoid activation function with 2 hidden layers, neurons split at 30 activation function tanh, and 100 training epochs. Performing better than the non-automized mode. Keeping the NAME column was crucial in achieving and going beyond the target.