Deep Learning Challenge

Overview:

A nonprofit foundation called Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. With our knowledge of machine learning and neural networks, using the features provided in the dataset we created a binary classifier that can predict whether 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 course of years. Within the dataset, there are several columns that capture metadata about each organization.

Results:

In order to begin with the data processing, we removed any information from the table that was not needed. For this data set, we dropped the EIN and NAME columns.

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

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSIDERATIONS	ASK_AMT	IS_SUCCESSFUL
0	T10	Independent	C1000	ProductDev	Association	1	0	N	5000	1
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	108590	1
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	0
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	6692	1
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	142590	1

The remainder columns were left as they would be considered features for this model. For binding purposes, the NAME column was added back in the second test. The data was then split for training and testing sets. Our target variable for the model was assigned as "IS_SUCCESSFUL". The data from the APPLICATION column was analyzed and the data from the CLASSIFICATION column was used for binning.

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

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

    # 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)
```

A variety of number ranges were used as a cutoff to bin "rare" variables together with the new value of "other" for each unique value. For categorical variables we used get_dummies( ) to encode after checking to see if the binning was successful.

Neural Network was applied on each model, a total of three layers. The three layer model generated 6,463 params on the first attempt and 43,251 on the second attempt. The number of hidden nodes were selected at random.

```
number_input_features = len( X_train_scaled[0])
hidden_nodes_layer1=100
hidden_nodes_layer2=30
hidden_nodes_layer3=10
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='sigmoid'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation='sigmoid'))

# 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)               39900

 dense_1 (Dense)             (None, 30)                3030

 dense_2 (Dense)             (None, 10)                310

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 43,251
Trainable params: 43,251
Non-trainable params: 0
```

```
number_input_features = len( X_train_scaled[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'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation='sigmoid'))

# 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)                4000

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

 dense_3 (Dense)             (None, 1)                 2

=================================================================
Total params: 6,463
Trainable params: 6,463
Non-trainable params: 0
```

Summary:

The first attempt using this model achieved an accuracy of about 73 percent (rounded to the nearest decimal). Whereas the second attempt when we used the “NAME” column, it achieved an accuracy of about 79 percent (rounded to the nearest decimal). This is a 6 percent difference between both attempts and about 4% over the target goal of 75 percent.

```
# 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: 0.5581 - accuracy: 0.7299 - 521ms/epoch - 2ms/step
Loss: 0.5581183433532715, Accuracy: 0.729912519454956
```

```
[23] # 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: 0.4906 - accuracy: 0.7858 - 495ms/epoch - 2ms/step
Loss: 0.4905915856361389, Accuracy: 0.7857726216316223
```

A recommendation for how a model could solve this classification problem is by adding multiple layers. This model is primary machine based therefore it teaches the computer to learn how to estimate and or classify the information given.