Neural Network Model Report

Data Preprocessing

Target Variables for module include:

- *APPLICATION_TYPE*
- *CLASSIFICATION*
- *IS_SUCCESSFUL*

```
dtype: int64
```

In [5]:
```python
# Look at APPLICATION_TYPE value counts for binning
app_count= application_df["APPLICATION_TYPE"].value_counts()
app_count
```

Out[5]:
```
T3     27037
T4      1542
T6      1216
T5      1173
T19     1065
T8       737
T7       725
T10      528
T9       156
T13       66
T12       27
T2        16
T25        3
T14        3
T29        2
T15        2
T17        1
Name: APPLICATION_TYPE, dtype: int64
```

In [6]:
```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 = app_count[app_count<500].index

# 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
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: APPLICATION_TYPE, dtype: int64
```

```
Other      276
Name: APPLICATION_TYPE, dtype: int64
```

In [7]:
```python
# Look at CLASSIFICATION value counts for binning
class_count = application_df["CLASSIFICATION"].value_counts()
class_count
```

Out[7]:
```
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
         ...
C4120        1
C8210        1
C2561        1
C4500        1
C2150        1
Name: CLASSIFICATION, Length: 71, dtype: int64
```

In [8]:
```python
# You may find it helpful to look at CLASSIFICATION value counts >1
class_count[class_count>1]
```

Out[8]:
```
C1000    17326
C2000     6074
C1200     4837
C3000     1918
C2100     1883
C7000      777
C1700      287
C4000      194
C5000      116
C1270      114
C2700      104
C2800       95
C7100       75
C1300       58
C1280       50
C1230       36
C1400       34
C7200       32
C2300       32
C1240       30
C8000       20
C7120       18
C1500       16
C1800       15
C6000       15
C1250       14
```

```
C1267        2
C1256        2
Name: CLASSIFICATION, dtype: int64
```

In [9]:
```python
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`
classifications_to_replace = class_count[class_count<1883].index

# 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[9]:
```
C1000    17326
C2000     6074
C1200     4837
Other     2261
C3000     1918
C2100     1883
Name: CLASSIFICATION, dtype: int64
```

In [10]:
```python
# Convert categorical data to numeric with `pd.get_dummies`
dummies_df =pd.get_dummies(application_df)
dummies_df.head()
```

Out[10]:

	STATUS	ASK_AMT	IS_SUCCESSFUL	APPLICATION_TYPE_Other	APPLICATION_TYPE_T10	APPLICATION_TYPE_T19	APPLICATION_TYPE_T
0	1	5000	1	0	1	0	
1	1	108590	1	0	0	0	
2	1	5000	0	0	0	0	
3	1	6692	1	0	0	0	
4	1	142590	1	0	0	0	

5 rows × 44 columns

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

# Split the preprocessed data into a training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state= 42)
```

In [12]:
```
# Create a StandardScaler instances
scaler = StandardScaler()

# Fit the StandardScaler
X_scaler = scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

Features of Model include:

In [4]:
```
# Determine the number of unique values in each column.
application_df.nunique()
```

Out[4]:
```
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
dtype: int64
```

Drop Non Beneficial Information:

- *EIN*
- *NAME*

In [3]:
```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(["EIN", "NAME"], axis = 1)
```

Compiling, Training, and Evaluating the Model

How many neurons, layers, and activation functions did you select for your neural network model, and why?

- *Measuring Accuracy*

Compile, Train and Evaluate the Model

n [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_layer_1= 80
hidden_layer_2 = 30

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer_1, activation="relu", input_dim = input_layer))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer_2, 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, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

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

n [14]:

```
# Compile the model
nn.compile(loss="binary_crossentropy", optimizer= "adam", metrics=["accuracy"])
```

Were you able to achieve the target model performance?

- *No, we reached at a 72.71% accuracy*

```
Trainable params: 5,981
Non-trainable params: 0
_________________________________________________________________
In [14]:
# Compile the model
nn.compile(loss="binary_crossentropy", optimizer= "adam", metrics=["accuracy"])

In [15]:
# Train the model
fit_model = nn.fit (X_train_scaled, y_train, epochs= 100)

Epoch 1/100
```

```
719/719 [==============================] - 1s 2ms/step - loss: 0.5334 - accuracy: 0.7409

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}")

354/354 - 1s - loss: 0.5608 - accuracy: 0.7272 - 583ms/epoch - 2ms/step
Loss: 0.5607906579971313, Accuracy: 0.7271843552589417

In [17]:
# Export our model to HDF5 file
nn.save('AlphabetSoupCharity.h5')
```

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

Optimization Attempt#1

- *Dropping more or fewer columns (dropping an additional column: organization).*
- *Increasing or decreasing the number of values for each bin(changed thresholds for application type in the classification column).*

```
In [2]:
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(["EIN", "NAME","ORGANIZATION"], axis = 1)
```

```
In [3]:
# Determine the number of unique values in each column.
application_df.nunique()
```

```
Out[3]: APPLICATION_TYPE            17
AFFILIATION                  6
CLASSIFICATION              71
USE_CASE                     5
STATUS                       2
INCOME_AMT                   9
SPECIAL_CONSIDERATIONS       2
ASK_AMT                   8747
IS_SUCCESSFUL                2
dtype: int64
```

```
Name: APPLICATION_TYPE, dtype: int64
```

```
In [5]:
# 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 = app_count[app_count<50].index

# 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[5]: T3       27037
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
T9         156
T13         66
Other       54
Name: APPLICATION_TYPE, dtype: int64
```

Optimization Attempt#2

- *Add more neurons to a hidden layer(adding more neurons to the 1st and 2nd hidden layers).*
- *Add more hidden layers(adding a 3rd hidden layer).*

In [16]:

```
# 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_layer_1= 100
hidden_layer_2 = 50
hidden_layer_3 = 20

nn2 = tf.keras.models.Sequential()

# First hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_layer_1, activation="relu", input_dim = input_layer))

# Second hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_layer_2, activation="relu"))

# Third hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_layer_3, activation="relu"))

# Output layer
nn2.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn2.summary()


# Compile the model
nn2.compile(loss="binary_crossentropy", optimizer= "adam", metrics=["accuracy"])


fit_model2 = nn2.fit (X_train_scaled, y_train, epochs= 100)

model_loss, model_accuracy = nn2.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

Optimization Attempt #3

- *Use different activation functions for the hidden layers(using sigmoid as activation function for all layers).*
- *Add or reduce the number of epochs to the training regimen(increasing epochs from 100 to 200).*

In [17]:
```
# 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_layer_1= 100
hidden_layer_2 = 50
hidden_layer_3 = 20

nn3 = tf.keras.models.Sequential()

# First hidden layer
nn3.add(tf.keras.layers.Dense(units=hidden_layer_1, activation="sigmoid", input_dim = input_layer))

# Second hidden layer
nn3.add(tf.keras.layers.Dense(units=hidden_layer_2, activation="sigmoid"))

# Third hidden layer
nn3.add(tf.keras.layers.Dense(units=hidden_layer_3, activation="sigmoid"))

# Output layer
nn3.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn3.summary()


# Compile the model
nn3.compile(loss="binary_crossentropy", optimizer= "adam", metrics=["accuracy"])


fit_model3 = nn3.fit (X_train_scaled, y_train, epochs= 200)

model_loss, model_accuracy = nn3.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 100)               5800

 dense_8 (Dense)             (None, 50)                5050

 dense_9 (Dense)             (None, 20)                1020

 dense_10 (Dense)            (None, 1)                 21

=================================================================
Total params: 11,891
Trainable params: 11,891
Non-trainable params: 0
```

Summary: *My second model with adding more neurons to a hidden layer(adding more neurons to the 1st and 2nd hidden layers),and adding more hidden layers(adding a 3rd hidden layer) had the best predictive accuracy score with a 72.89.*