Analysis

Overview

The purpose of this analysis is to determine which applicants will have the highest chances of being successful should they be selected through the Alphabet Soup foundation. Alphabet Soup foundation will create an algorithm through the use of machine learning, neural networks, and the provided csv files of all the organizations.

Data Preprocessing

- What variable(s) are the target(s) for your model?
  The target variable is the 'IS_SUCCESSFUL' column.

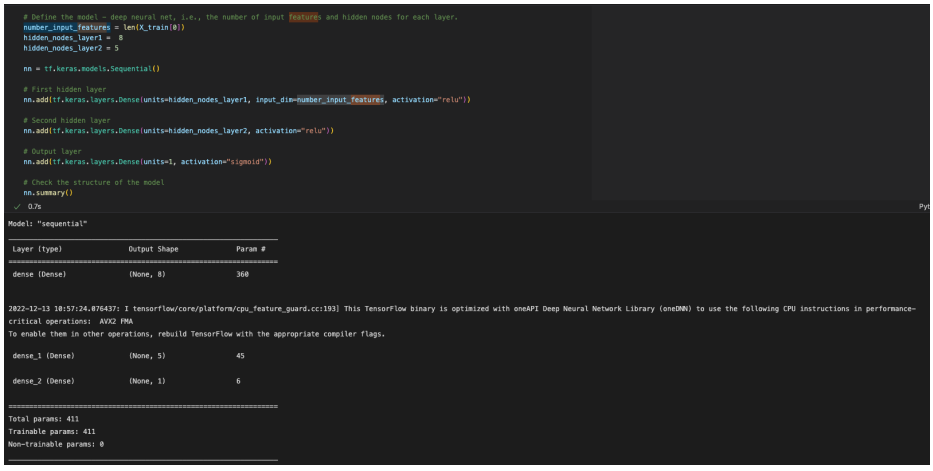
- What variable(s) are the features for your model?
  The features of the model include the rest of the columns in the dataframe after dropping the 'EIN' and 'NAME' columns.

- What variable(s) should be removed from the input data because they are neither targets nor features?
  Removed the 'EIN' and 'NAME' columns from the dataframe. Also, replaced the 'CLASSIFICATION' column with 'Other'.

Compiling, Training, and Evaluating the Model

- How many neurons, layers, and activation functions did you select for your neural network model, and why?
  Here is the model for my first attempt:
  Layer1 = 8 : activation function = relu
  Layer2 = 5 : activation function = relu

  Loss: 0.5540196299552917, Accuracy: 0.7240816354751587



```
# 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 =  8
hidden_nodes_layer2 = 5

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()
```

```
0.7s
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 360


2022-12-13 10:57:24.076437: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-
critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

 dense_1 (Dense)             (None, 5)                 45

 dense_2 (Dense)             (None, 1)                 6

=================================================================
Total params: 411
Trainable params: 411
Non-trainable params: 0
_________________________________________________________________
```

- Were you able to achieve the target model performance?
  No, I was unable to reach the target model performance. I was only able to achieve 0.7240816354751587 accuracy.

```
# 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}")
```

[23] ✓ 0.9s

```
268/268 - 1s - loss: 0.5540 - accuracy: 0.7241 - 695ms/epoch - 3ms/step
Loss: 0.5540196299552917, Accuracy: 0.7240816354751587
```

- What steps did you take in your attempts to increase model performance?
  During my second and third attempts I tried to increase the hidden layers and also test out a different activation for hidden layers.

Attempt # 2

```
# 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 =  7
hidden_nodes_layer2 = 14
hidden_nodes_layer3 = 20

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()

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

✓ 0.3s

```
Output exceeds the size limit. Open the full output data in a text editor
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 315

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 20)                300

 dense_3 (Dense)             (None, 1)                 21

=================================================================
Total params: 748
Trainable params: 748
Non-trainable params: 0
_________________________________________________________________
Model: "sequential"
```

```
# 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}")
```

✓ 1.3s

```
268/268 - 1s - loss: 0.5506 - accuracy: 0.7271 - 666ms/epoch - 2ms/step
Loss: 0.550557553768158, Accuracy: 0.7271137237548828
```

Attempt #3

Third Attempt

```
# 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 = 20
hidden_nodes_layer2 = 40
hidden_nodes_layer3 = 40
hidden_nodes_layer4 = 50

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="tanh"))

# Fourth hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer4, activation="relu"))

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

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

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

✓ 0.4s

```
Output exceeds the size limit. Open the full output data in a text editor
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 20)                900

 dense_10 (Dense)            (None, 40)                840

 dense_11 (Dense)            (None, 40)                1640

 dense_12 (Dense)            (None, 50)                2050

 dense_13 (Dense)            (None, 1)                 51

=================================================================
Total params: 5,481
Trainable params: 5,481
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 20)                900

...
Total params: 5,481
Trainable params: 5,481
Non-trainable params: 0
```

```
# 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}")
```

✓ 1.1s

```
268/268 - 1s - loss: 0.5589 - accuracy: 0.7251 - 822ms/epoch - 3ms/step
Loss: 0.5589492321014404, Accuracy: 0.7251312136650085
```

Summary

Based on the results, it seems that there is a loss of 0.55 meaning that the model can be further optimized. I was unable to achieve more than 75% accuracy, I believe to hit this goal there would need to be multiple layers and nodes to achieve this. Also, to choose the optimal number of layers and nodes to avoid overfitting. I was unable to find the correct number so I can only recommend trying using a higher node unit and also run longer epochs.