Deep Learning Challenge

Overview:

This deep learning challenge is for a non-profit foundation, Alphabet Soup, to determine to which applicants they should provide the funding. The CSV file contains historical data of organizations which have received fundings from Alphabet Soup, including name of the organizations, application type, affiliated sector of industry, government organization classification, use case for funding, organization type, active status, income classification, any special aspects to consider for application, funding amount, and finally whether the applicant was successful or not. With machine learning and neural networks, we set up a model and made it to predict future applicants.

Data Processing & Results:

First model, saved under AlphabetSoupCharity.ipynb file, shows a binary classification model to predict which organization will be successful. In order to do this, we first dropped 'EIN' and 'NAME' columns. Next, we have created binning for application type and classification to prepare the data and make the "rare" categorical variables into a new value "other." Then we separated the target array, y, which was the column 'IS_SUCCESSFUL' in this case, and the features array, X. After we have trained and tested the features, we compiled and trained the model. In order to do this, we had two hidden layers, with 80 and 30 neurons, respectively, and used the "relu" function. Then we had the "sigmoid" function with 1 node as the outer layer to create binary classifier model. As a result, the accuracy of this model came out to be 72.5%.

In order to increase the accuracy to be over 75%, there were several trials and errors I had to go through. First, I tried increasing the epochs from 100 to 200, but the accuracy did not change much and remained at 72.58%.

```
Epoch 199/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5333 - accuracy: 0.7425
Epoch 200/200
804/804 [==============================] - 2s 2ms/step - loss: 0.5336 - accuracy: 0.7420
```

```
] # 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: 0.5653 - accuracy: 0.7258 - 501ms/epoch - 2ms/step
Loss: 0.5652855038642883, Accuracy: 0.7258309125900269
```

Then, I tried to put a third layer with “sigmoid” function, right before the output layer to see if it improves the accuracy but it did not, and still remained at 72.5%.

```
# 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_scaled[0])
hidden_nodes_layer1 =  80
hidden_nodes_layer2 = 30
hidden_nodes_layer3 = 10
hidden_nodes_layer4 = 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=hidden_nodes_layer4, 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, 10)                310

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 6,271
Trainable params: 6,271
Non-trainable params: 0
```

```
Epoch 28/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5408 - accuracy: 0.7389
Epoch 29/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5405 - accuracy: 0.7390
```

```
[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 - loss: 0.5538 - accuracy: 0.7251 - 550ms/epoch - 2ms/step
Loss: 0.553816020488739, Accuracy: 0.7251312136650085
```

Next, I went back to the top of the jupyter notebook, and only dropped the 'EIN' column and left 'NAME' column active to create more binning. Initially, I had set 400 as the number of values for NAME binning, but the accuracy remained same at 72.5% (the work saved under AlphabetSoupCharity_Optimization.ipynb).

```
# 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: 0.5428 - accuracy: 0.7255 - 518ms/epoch - 2ms/step
Loss: 0.5428326725959778, Accuracy: 0.7254810333251953
```

Finally, I changed the number of values for NAME binning to 100, changed the hidden nodes to 10 and 5, respectively, the accuracy rose upto 75.1% (saved under AlphabetSoupCharity_Optimization_2.ipynb).

```
# 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: 0.4982 - accuracy: 0.7515 - 544ms/epoch - 2ms/step
Loss: 0.49819475412368774, Accuracy: 0.7514868974685669
```