Module 21. Deep Learning

Charity Data

Overview
The goal of this project is to create an algorithm using machine learning and neural networks to predict whether applicants will be successful.

Process
I was given a CSV file that I read into Pandas. This file contained more than 34,000 organizations that have received funding from the fictional foundation along with several columns of metadata about each organization.

PREPROCESSING

I pre-processed the data by:

- Dropping non-beneficial columns
- Finding the number of data points for each unique value for each of the columns that had more than 10 unique values - APPLICATION_TYPE and CLASSIFICATION
- Choosing a cutoff point of 600 and 800, respectively, to bin rare Categorical values together into a new value called "Other",
- Using `pd.get_dummies()` to convert categorical data to numeric, dividing the data into a target array (IS_SUCCESSFUL) and features arrays,
- applying the `train_test_split` to create a testing and a training dataset
- Using `StandardScaler` to scale the training and testing sets

The target variable (y) was IS_SUCCESSFUL. The data was split into training and test subsets.

COMPILING, TRAINING, AND EVALUATING THE MODEL
The model was required to achieve a target predictive accuracy higher than 75%. I made three official attempts using machine learning and neural networks. They all resulted in the same accuracy rate – around 73%.

Results from each model attempt are detailed below:

ATTEMPT #1
The first attempt (Resources/AlphabetSoupCharity.h5) resulted in an accuracy score of 72.65%. This means that 72.65% of the model's predicted values align with the dataset's true values.

```
# 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 = 9
hidden_nodes_layer2 = 18


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

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

The hyperparameters used were:
• layers = 2
o layer1 = 9 neurons and ‘relu’ activation function
o layer2 = 18 neurons and ‘relu’ activation function
• epochs = 100

```
[41] # 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: 0.5535 - accuracy: 0.7265 - 454ms/epoch - 2ms/step
     Loss: 0.5534908175468445, Accuracy: 0.7265306115150452
```

ATTEMPT #2
For my second attempt (Resources/AlphabetSoupCharity_Optimization.h5) I added another layer. This attempt resulted in an accuracy score of 72.75%. This means that 72.75% of the model’s predicted values align with the dataset’s true values.

```
#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 = 9
hidden_nodes_layer2 = 18
hidden_nodes_layer3 = 27

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()
```

The hyperparameters used were:
• layers = 3
o layer1 = 9 neurons : activation function = ‘relu’
o layer2 = 18 neurons : activation function = ‘relu’
o layer3 = 27 neurons : activation function = ‘relu’

• epochs = 100

```
[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: 0.5559 - accuracy: 0.7276 - 501ms/epoch - 2ms/step
Loss: 0.5559346675872803, Accuracy: 0.727580189704895
```

ATTEMPT #3

For my third and final attempt (Resources/AlphabetSoupCharity_Optimization_2.h5) I kept the third layer and changed the activation function for layers 2 and 3. This attempt resulted in an accuracy score of 72.62%. This means that 72.62% of the model's predicted values align with the dataset's true values.

```
# 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 = 9
hidden_nodes_layer2 = 18
hidden_nodes_layer3 = 27

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"))

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

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

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

The hyperparameters used were:

• layers = 3

o layer1 = 9 neurons : activation function = 'relu'

o layer2 = 18 neurons : activation function = 'tanh'

o layer3 = 27 neurons : activation function = 'tanh'

• epochs = 100

```
# 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: 0.5537 - accuracy: 0.7263 - 472ms/epoch - 2ms/step
Loss: 0.5537424683570862, Accuracy: 0.7262973785400391
```

Summary

In the three attempts I made, the model was unable to achieve a target predictive accuracy higher than 72.75%. Hypertuning resulted in virtually no improvement. I would consider using another classification model to see if it is better at predicting whether applicants will be successful.