Lennox Nguyen
11/16/2022

Deep Learning Project: Charity Funding Predictor

Overview:

The non-profit foundation Alphabet Soup wants to create an algorithm to predict whether or not applicants for funding will be successful. With knowledge of machine learning and neural networks, we must use the features in the provided dataset to create a binary classifier that is capable of predicting whether applicants will be successful if funded by Alphabet Soup.

Results:

To begin the data processing, we removed any irrelevant data. After dropping the EIN and NAME columns, the remaining columns were considered to be features for the model. Although the NAME column was added back in the second test, the CLASSIFICATION and APPLICATION_TYPE columns were replaced with the OTHER column, due to high fluctuation. Then, the data was split into training and testing sets of data. The target variable for the model was the IS_SUCCESSFUL column, verified by the value with 1 being yes and 0 being no. APPLICATION column's data was analyzed while the CLASSIFICATION column's value was used for binning. Several data points were used as a cutoff to bin rare variables together with the value of the OTHER column for each unique value. Categorical variables were encoded by get_dummies() after checking to see if the binning was successful.

For the Compiling, Training, and Evaluating the Model section, there were three layers total for each model after applying Neural Networks. The number of hidden nodes were dictated by the number of features.

In [12]:
```
# 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 = 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'))

# 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

=================================================================
Total params: 6,461
Trainable params: 6,461
Non-trainable params: 0
```

6461 parameters were created by a three-layer training model. The first attempt was just over 72% accuracy, which was under a desired 75% accuracy, but not so far off.

In [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 - 0s - loss: 0.5621 - accuracy: 0.7292 - 151ms/epoch - 565us/step
Loss: 0.5620855093002319, Accuracy: 0.7292128205299377
```

For the optimization, the second attempt with the NAME column in the dataset, achieved an accuracy of just below 79%, which was about 4% over target. A total of 39,341 parameters.

In [12]:
```
# 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 = 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'))

# 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)                36880

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 39,341
Trainable params: 39,341
Non-trainable params: 0
```

In [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 - 0s - loss: 0.4895 - accuracy: 0.7867 - 157ms/epoch - 586us/step
Loss: 0.48945745825767517, Accuracy: 0.7867055535316467
```