Deep Learning Challenge: Charity Funding Predictor

Overview:

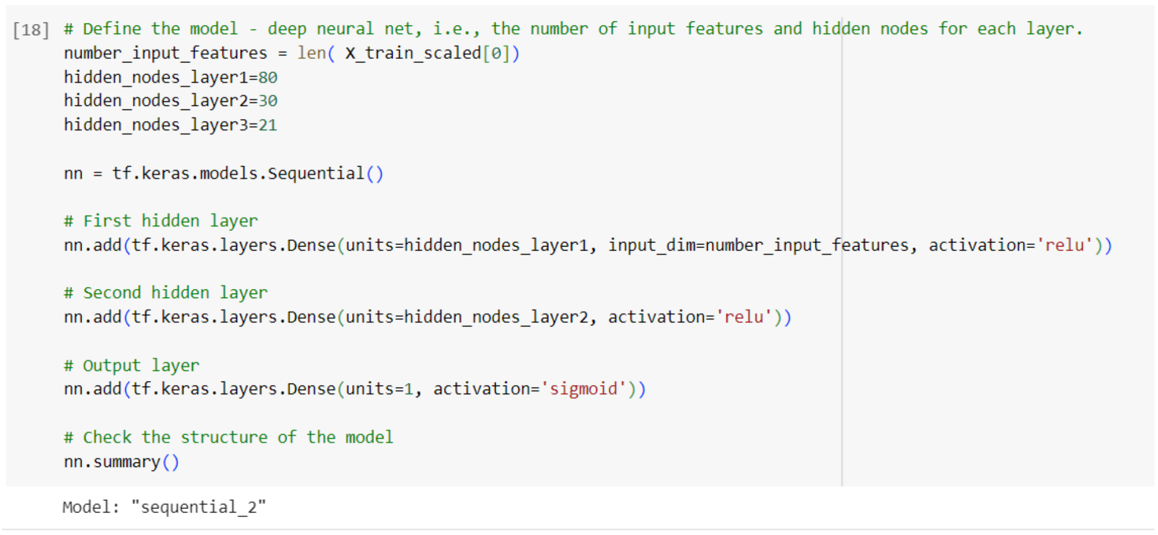
The non-profit foundation Alphabet Soup wants to create an algorithm to predict whether or not applicants for funding will be successful. With knowledge of machine learning and neural networks, we must use the features in the provided dataset to create a binary classifier that is capable of predicting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

Results:

To begin the data processing we removed any irrelevant information. After dropping EIN and NAME the remaining columns were to be considered features for the model. Name was added back into the second test for binning purposes. The data was then split for training and testing sets. The target variable for the model was labeled "IS_SUCCESSFUL" and has the value of 1 for yes and 0 for no. APPLICATION data was analyzed and "CLASSIFICATION value was used for binning. We used several data points as a cutoff to bin "rare" variables together with the new value of "Other" for each unique value. Categorical variables were encoded by get_dummies() after checking to see if the binning was successful.

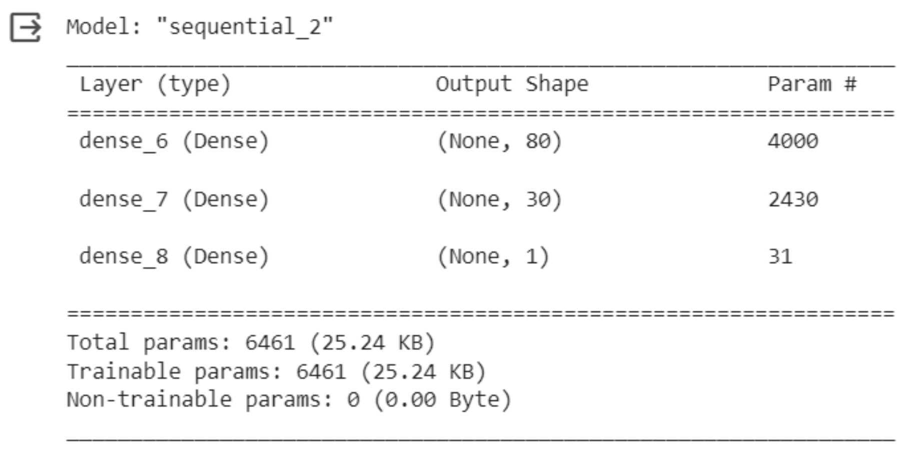
Compiling, Training, and Evaluating the Model:

There were three layers total for each model after applying Neural Networks. The number of hidden nodes were dictated by the number of features.

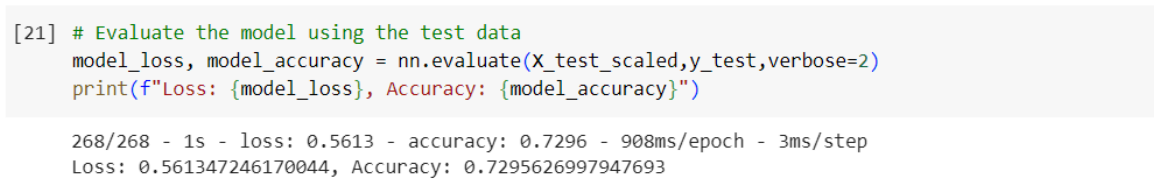


```
[18] # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
     number_input_features = len( X_train_scaled[0])
     hidden_nodes_layer1=80
     hidden_nodes_layer2=30
     hidden_nodes_layer3=21

     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_2"
```

477 parameters were created by a three-layer training model. The first attempt was just over 73% accuracy which was under a desired 75% but not too far off.



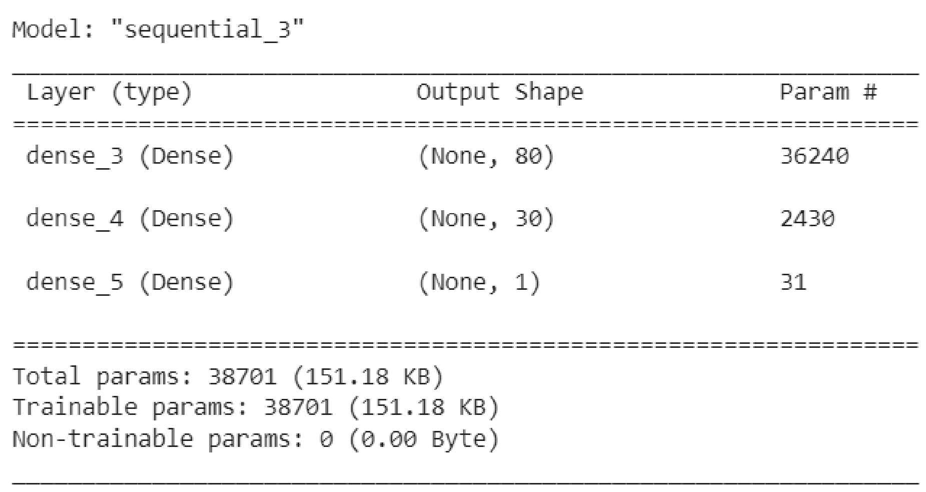
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 80)                4000

 dense_7 (Dense)             (None, 30)                2430

 dense_8 (Dense)             (None, 1)                 31

=================================================================
Total params: 6461 (25.24 KB)
Trainable params: 6461 (25.24 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



```
[21] # 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: 0.5613 - accuracy: 0.7296 - 908ms/epoch - 3ms/step
     Loss: 0.561347246170044, Accuracy: 0.7295626997947693
```

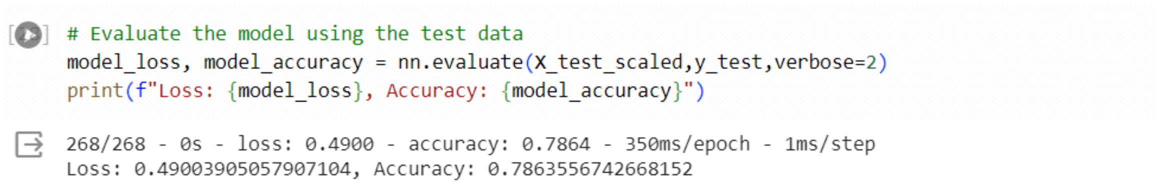
Optimization:

The second attempt with the “NAME” column in the dataset, achieved an accuracy of almost 79%. This is 4% over the target 75% with 3,298 parameters.



```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 80)                36240

 dense_4 (Dense)             (None, 30)                2430

 dense_5 (Dense)             (None, 1)                 31

=================================================================
Total params: 38701 (151.18 KB)
Trainable params: 38701 (151.18 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Multiple layers should be used for deep learning models since it learns how to predict and classify information based on computer filtering inputs through layers.



```
# 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: 0.4900 - accuracy: 0.7864 - 350ms/epoch - 1ms/step
Loss: 0.49003905057907104, Accuracy: 0.7863556742668152
```