Analysis

Deep Learning Challenge

Overview:

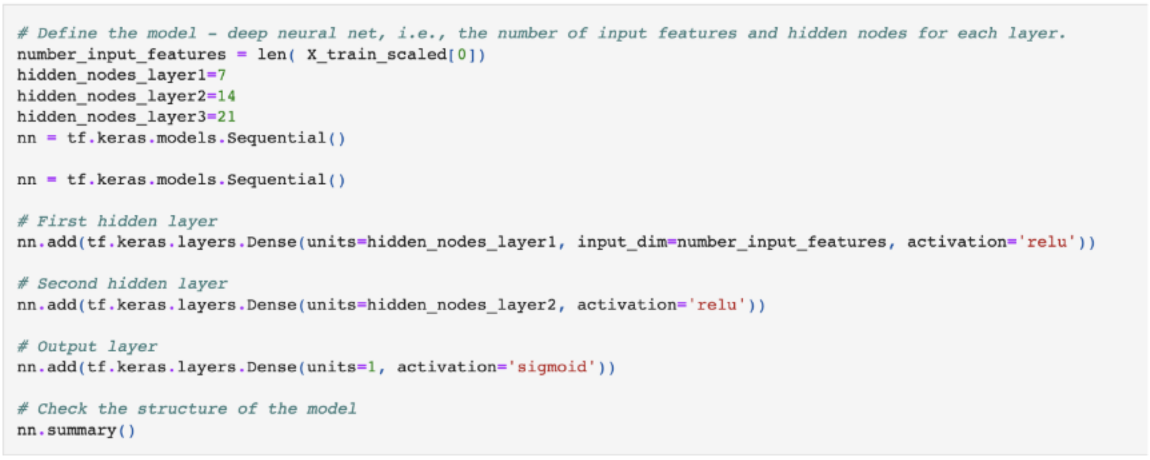
The non-profit foundation Alphabet Soup wants to create an algorithm to predict whether applicants for funding will be successful. With knowledge of machine learning and neural networks, we must use the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

Results:

For the data pre-processing and cleaning, we removed information that is irrelevant. After dropping NAME and EIN the remaining columns were considered for the model features. For the binning purposes NAME was added back into the second test and then used to split for training and testing sets. The target variable for the model was labelled “IS_SUCCESSFUL” and 1 was assigned for the value of yes and 0 for no. APPLICATION data was analysed and “CLASSIFICATION value was used for binning. We used several data points as a cut off 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.

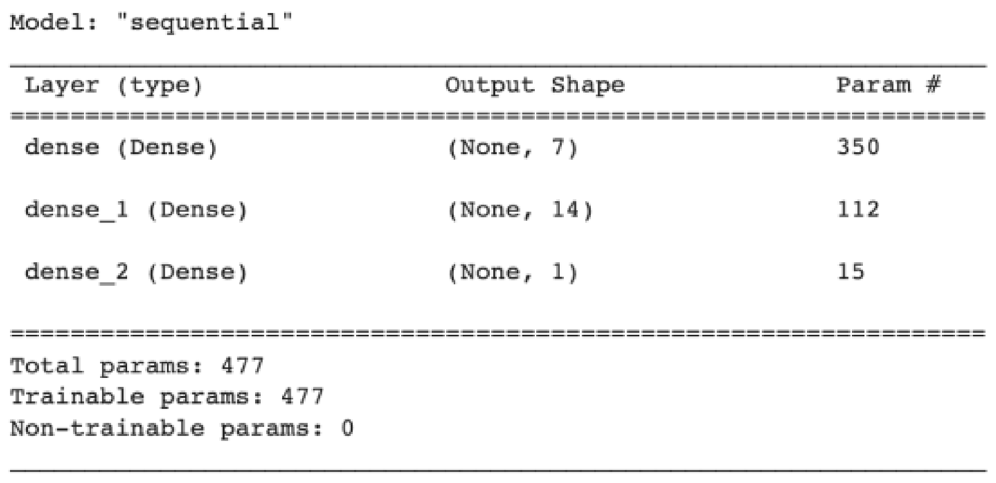
Compiling, Training, and Evaluating the Model:

After applying Neural Networks, there were three layers total for each model.

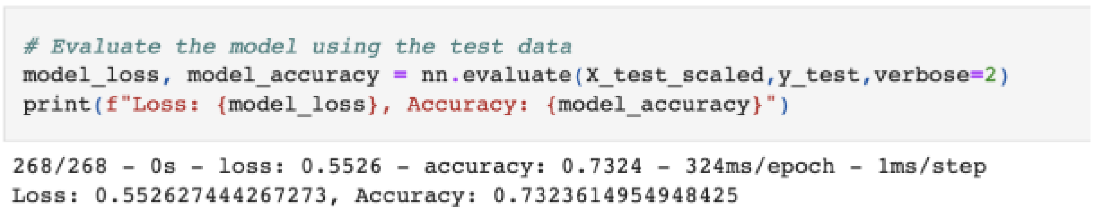


```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len( X_train_scaled[0])
hidden_nodes_layer1=7
hidden_nodes_layer2=14
hidden_nodes_layer3=21
nn = tf.keras.models.Sequential()

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()
```

The number of hidden nodes were dictated by the number of features. A three-layer training model created 477 parameters. The first attempt showed over 73% accuracy though it was under the desired 75% but is not too far off.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 350

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 477
Trainable params: 477
Non-trainable params: 0
_________________________________________________________________
```

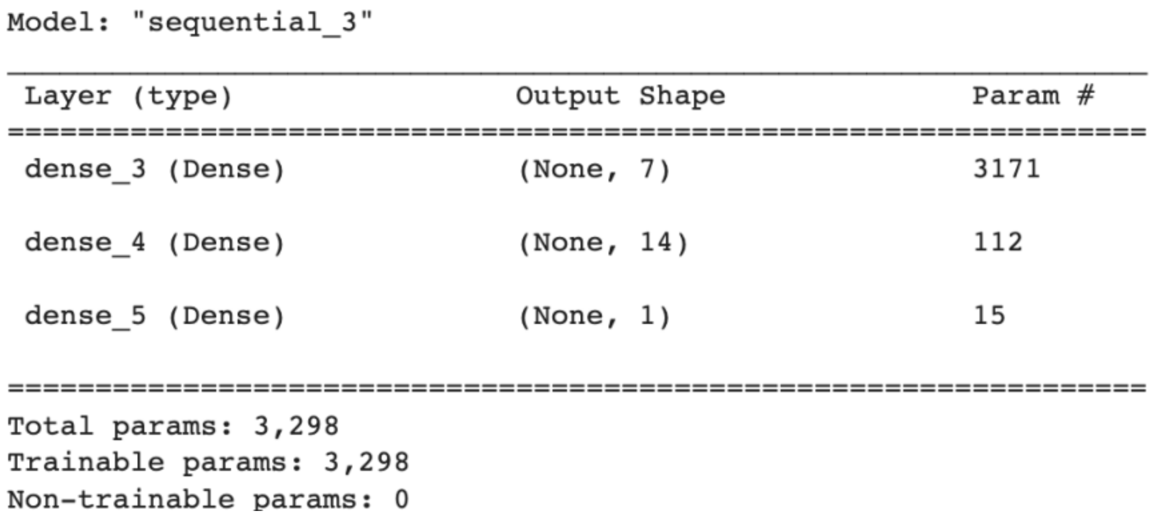


```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5526 - accuracy: 0.7324 - 324ms/epoch - 1ms/step
Loss: 0.552627444267273, Accuracy: 0.7323614954948425
```

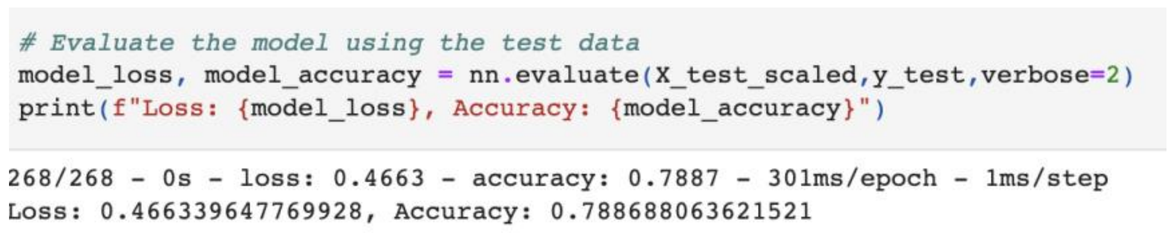
Optimization:

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



```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 7)                 3171

 dense_4 (Dense)             (None, 14)                112

 dense_5 (Dense)             (None, 1)                 15

=================================================================
Total params: 3,298
Trainable params: 3,298
Non-trainable params: 0
```

Using multiple layers increases the accuracy of the prediction and should be used for deep learning models since it learns how to predict and classify information based on computer filtering inputs through the layers.



```python
# 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: 0.4663 - accuracy: 0.7887 - 301ms/epoch - 1ms/step
Loss: 0.466339647769928, Accuracy: 0.788688063621521
```