THE UNIVERSITY OF BIRMINGHAM

Bootcamp Data analytics

Module 21- Deep learning Challenge

Date of report/submission: 7th of Jul. 2023

Lecturer: Eric and Kristina

by Alemseghed Ghebrezghi

EMAIL: AT.GHEBREZGHI@GMAIL.COM

CONTENTS

1. Overview

The Alphabet Soup Nonprofit Foundation sought a binary classifier in order to facilitate the selection of applicants for funding. The binary classifier is designed to assess the likelihood of successful applicants being funded by Alphabet Soup.

2. Data processing

EIN and Name were eliminated from the dataset, and all remaining metrics were incorporated into the model. Classification and Application Type are features of the model. The following points have been considered in the data processing.

- What variable(s) are the target(s) for your model?
- What variable(s) are the features for your model?
- What variable(s) should be removed from the input data because they are neither targets nor features?

2.1. Compiling, Training, and Evaluating the Model:

Neural Network was used on each model and originally set with 2 hidden layers for first and second attempt using relu function (activation function) to create a threshold and sigmoid function (activation function) for output layer. For the final model(the third attempt), 3 hidden layers have been used with some changes in number of nodes to acquire the required accuracy. Hence, the code for defining the models (attempts) have been shown in the picture below.

- First attempt

Compile, Train and Evaluate the Model

In [12]:
```
# (FIRST ATTEMPT - ACCURACY 74%)
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features_total = len(X_train[0])
hidden_nodes_layer1 = 7
hidden_nodes_layer2 = 4
number_of_neurons = tf.keras.models.Sequential()

# First hidden layer
number_of_neurons.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim = input_features_total, activation = "relu")

# Second hidden layer
number_of_neurons.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation = "relu"))

# Output layer
number_of_neurons.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))
# Check the structure of the model
number_of_neurons.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 137284

 dense_1 (Dense)             (None, 4)                 32

 dense_2 (Dense)             (None, 1)                 5

=================================================================
Total params: 137,321
Trainable params: 137,321
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```
# Compile the model(FIRST ATTEMPT)
number_of_neurons.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

In [14]:
```
# Train the model(FIRST ATTEMPT)
fit_model = number_of_neurons.fit(X_train_scaled,y_train,epochs=100)
```

```
804/804 [==============================] - 4s 5ms/step - loss: 0.0862 - accuracy: 0.9633
Epoch 92/100
804/804 [==============================] - 4s 5ms/step - loss: 0.0862 - accuracy: 0.9635
Epoch 93/100
804/804 [==============================] - 3s 4ms/step - loss: 0.0860 - accuracy: 0.9633
Epoch 94/100
```

In [15]:
```
# Evaluate the model using the test data(FIRST ATTEMPT)
model_loss, model_accuracy = number_of_neurons.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.5857 - accuracy: 0.7418 - 1s/epoch - 4ms/step
Loss: 0.5857194066047668, Accuracy: 0.7418075799942017
```

➢ Second attempt

In [16]:
```
# (SECOND ATTEMPT - ACCURACY 57%)
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features_total = len(X_train[0])
hidden_nodes_layer1 =10
hidden_nodes_layer2 = 8
number_of_neurons = tf.keras.models.Sequential()

# First hidden layer
number_of_neurons.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim = input_features_total, activation = "relu"))

# Second hidden layer
number_of_neurons.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation = "relu"))

# Output layer
number_of_neurons.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))
# Check the structure of the model
number_of_neurons.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 10)                196120

 dense_4 (Dense)             (None, 8)                 88

 dense_5 (Dense)             (None, 1)                 9

=================================================================
Total params: 196,217
Trainable params: 196,217
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
# Compile the model(SECOND ATTEMPT)
number_of_neurons.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

In [18]:
```
# Train the model(SECOND ATTEMPT)
fit_model = number_of_neurons.fit(X_train_scaled,y_train,epochs=100)
```

```
Epoch 74/100
804/804 [==============================] - 4s 5ms/step - loss: 0.0839 - accuracy: 0.9655
Epoch 75/100
804/804 [==============================] - 4s 5ms/step - loss: 0.0837 - accuracy: 0.9658
Epoch 76/100
804/804 [==============================] - 4s 5ms/step - loss: 0.0838 - accuracy: 0.9651
Epoch 77/100
```

In [19]:
```
# Evaluate the model using the test data(SECOND ATTEMPT)
model_loss, model_accuracy = number_of_neurons.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.9503 - accuracy: 0.5698 - 940ms/epoch - 4ms/step
Loss: 0.9502856731414795, Accuracy: 0.5697959065437317
```

- **Third attempt**

```
In [20]:
# (THIRD ATTEMPT - ACCURACY 76%)
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features_total = len(X_train[0])
hidden_nodes_layer1 = 7
hidden_nodes_layer2 = 14
hidden_nodes_layer3 = 21
number_of_neurons = tf.keras.models.Sequential()

# First hidden layer
number_of_neurons.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim = input_features_total, activation = "relu")

# Second hidden layer
number_of_neurons.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation = "relu"))

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

# Output layer
number_of_neurons.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))
# Check the structure of the model
number_of_neurons.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 7)                 137284

 dense_7 (Dense)             (None, 14)                112

 dense_8 (Dense)             (None, 21)                315

 dense_9 (Dense)             (None, 1)                 22

=================================================================
Total params: 137,733
Trainable params: 137,733
Non-trainable params: 0
_________________________________________________________________
```

```
In [21]:
# Compile the model(THIRD ATTEMPT)
number_of_neurons.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

```
In [22]:
# Train the model(THIRD ATTEMPT ATTEMPT)
fit_model = number_of_neurons.fit(X_train_scaled,y_train,epochs=100)
```

```
804/804 [==============================] - 4s 5ms/step - loss: 0.0910 - accuracy: 0.9637
Epoch 92/100
804/804 [==============================] - 4s 5ms/step - loss: 0.0874 - accuracy: 0.9648
```

```
In [23]:
# Evaluate the model using the test data(THIRD ATTEMPT)
model_loss, model_accuracy = number_of_neurons.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.5962 - accuracy: 0.7636 - 889ms/epoch - 3ms/step
Loss: 0.5962450504302979, Accuracy: 0.7636151313781738
```

3. . Summary

From the deep learning models the accuracy has been found to be 74, 57 and 76% for first, second and third attempts respectively. In terms of assessment of each model, the second model is far away from the target which is 75%. Though the first attempt (74%) is close to the target it does not meet the requirement. However, the third attempt's result at least good enough to hit the target and hence it can be taken as a final model. For the model to continue to forecast with optimum accuracy, it is necessary to consider multiple hidden layers with an increase of number of nodes(neurons or preceptors) and needs removal of features that dos does not affect the result.