Tadeo Espinoza

Module 21 Challenge Report

Overview:

- Alphabet Soup, a nonprofit foundation, needs a solution to improve its funding allocation process. Using our skills in machine learning and neural networks, our task is to develop an efficient binary classifier. This classifier will examine a comprehensive dataset containing details about more than 34,000 organizations that have received funding from Alphabet Soup over the years. The main goal is to develop a predictive tool that helps the foundation's business team in pinpointing and backing applicants who are most likely to succeed in their endeavors.

Results:

Data Preprocessing

- Target Variable: "IS_SUCESSFUL"
- Feature Variables: "INCOME_AMT" and "ASK_AMT", "Classification" and "Application Type" as well.
- Variables removed: "EIN" and "Name"

In [2]:

```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df =application_df.drop(columns = ['EIN', 'NAME'])
application_df
```

Out[2]:

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSID
0	T10	Independent	C1000	ProductDev	Association	1	0	
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	
...	...	...	...	...	...	...	...	
34294	T4	Independent	C1000	ProductDev	Association	1	0	
34295	T4	CompanySponsored	C3000	ProductDev	Association	1	0	
34296	T3	CompanySponsored	C2000	Preservation	Association	1	0	
34297	T5	Independent	C3000	ProductDev	Association	1	0	
34298	T3	Independent	C1000	Preservation	Co-operative	1	1M-5M	

34299 rows × 10 columns

- **How many neurons, layers, and activation functions did you select for your neural network model, and why?**

- For my model, I used two hidden layers and an output layer. For this model, the neurons for the layers were 10 and 20 to keep it small and simple. “Relu” function was used for these, and a “sigmoid” function was used for the outer layer, which only had 1 neuron.

```
In [13]:
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 10
hidden_nodes_layer2 = 20
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 10)                460

 dense_4 (Dense)             (None, 20)                220

 dense_5 (Dense)             (None, 1)                 21

=================================================================
Total params: 701 (2.74 KB)
Trainable params: 701 (2.74 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- **Were you able to achieve the target model performance?**

- Accuracy: 72.9%, no. Target accuracy was 75%.

- **What steps did you take in your attempts to increase model performance?**

- I had three different attempts at trying to increase the model’s performance. All three attempts focused specifically on the number of layers and the number of neurons each layer had.
- **First attempt**: added additional hidden layer (3 total) and increased neuron counts for each layer: 85, 95, 105. 72.80% accuracy.
- **Second attempt**: added additional layer (3 total) and this time decreasing neuron counts significantly. (4, 8, 12). 72.84% accuracy.

- **Third attempt**: added TWO additional hidden layers (4 TOTAL) and kept the same neuron amount for all 4 layers. (20 each). Accuracy: 72.63%.

Attempt 1:

In [12]:
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 85
hidden_nodes_layer2 = 95
hidden_nodes_layer3 = 105
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=input_features, activation='relu'))
# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))
# Third 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()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 85)                3825

 dense_1 (Dense)             (None, 95)                8170

 dense_2 (Dense)             (None, 105)               10080

 dense_3 (Dense)             (None, 1)                 106

=================================================================
Total params: 22181 (86.64 KB)
Trainable params: 22181 (86.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Attempt 2:

In [12]:
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 4
hidden_nodes_layer2 = 8
hidden_nodes_layer3 = 12
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=input_features, activation='relu'))
# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))
# Third 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()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4)                 180

 dense_1 (Dense)             (None, 8)                 40

 dense_2 (Dense)             (None, 12)                108

 dense_3 (Dense)             (None, 1)                 13

=================================================================
Total params: 341 (1.33 KB)
Trainable params: 341 (1.33 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Attempt 3:

In [32]:
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 20
hidden_nodes_layer2 = 20
hidden_nodes_layer3 = 20
hidden_nodes_layer4 = 20
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=input_features, activation='relu'))
# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))
# Third layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation='relu'))
# Fourth layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer4, 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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 20)                900

 dense_10 (Dense)            (None, 20)                420

 dense_11 (Dense)            (None, 20)                420

 dense_12 (Dense)            (None, 20)                420

 dense_13 (Dense)            (None, 1)                 21

=================================================================
Total params: 2181 (8.52 KB)
Trainable params: 2181 (8.52 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Summary:

It seems that after our original model, the three attempts at improving model performance based on changing number of layers and neurons did not really produce that much of a difference.

Original: 72.89%

1st attempt: 72.80%

2nd attempt: 72.84%

3rd attempt: 72.63%