Deep Learning Model Performance Report

Nandhini Nallathambi | Data Analytics Boot Camp | 12th February 2023

Table of Contents

Overview

The aim of this challenge is to create a tool for a non-profit foundation (Alphabet Soup) select applicants for funding with the best chance of success in their ventures.

The source CSV file contains more than 34,000 organizations that have received funding from Alphabet Soup over the years. The dataset contains the following columns that capture metadata about each organization, such as:

- **EIN** and **NAME**—Identification columns
- **APPLICATION_TYPE**—Alphabet Soup application type
- **AFFILIATION**—Affiliated sector of industry
- **CLASSIFICATION**—Government organization classification
- **USE_CASE**—Use case for funding
- **ORGANIZATION**—Organization type
- **STATUS**—Active status
- **INCOME_AMT**—Income classification
- **SPECIAL_CONSIDERATIONS**—Special considerations for application
- **ASK_AMT**—Funding amount requested
- **IS_SUCCESSFUL**—Was the money used effectively

Using machine learning and neural networks, the features in dataset above are used to create a binary classifier that can predict (higher than 75% accuracy) whether applicants will be successful if funded by Alphabet Soup.

Data Pre-Processing

- **Target**: The target for the model is the column **IS_SUCESSFUL** which contains the values 0 and 1 to indicate if the money was used effectively.
- **Features**: The features used in the model are the columns listed in the image below:

```
# Determine the number of unique values in each column.
application_df.nunique()
```

```
APPLICATION_TYPE            17
AFFILIATION                  6
CLASSIFICATION              71
USE_CASE                     5
ORGANIZATION                 4
STATUS                       2
INCOME_AMT                   9
SPECIAL_CONSIDERATIONS       2
ASK_AMT                   8747
IS_SUCCESSFUL                2
dtype: int64
```

- **Variables Removed**: The variables EIN and NAME were removed from the input data as they are neither targets nor features.
- The categorical data were converted to numeric, and the resulting dataset had 44 columns.

```
# Convert categorical data to numeric with `pd.get_dummies`
application_numeric = pd.get_dummies(application_df)
application_numeric.head()
```

	STATUS	ASK_AMT	IS_SUCCESSFUL	APPLICATION_TYPE_Other	APPLICATION_TYPE_T10	APP
0	1	5000	1	0	1	
1	1	108590	1	0	0	
2	1	5000	0	0	0	
3	1	6692	1	0	0	
4	1	142590	1	0	0	

5 rows × 44 columns

Compiling, Training and Evaluating the Model

Keras Sequential Model

- **Input Features**: The size of the input data selected is the number of input features for the model. Here it is 43.
- **Output Layer Size**: The Output layer size is 1, as the target column is binary here.
- **Hidden Nodes**:
    - The number of hidden nodes in Layer 1 is 85, considering a rule of thumb that the number of hidden neurons must be less than twice the size of the input layer. (Krishnan, 2021)
    - The number of hidden nodes in Layer 2 is 30, considering a rule of thumb that the number of hidden neurons must be 2/3 the size of the input layer, plus the size of the output layer (Krishnan, 2021)

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.

# Size of data is selected as the number of input features here
input_features = X.shape[1]
print("Input Features:",input_features)

# Size of the output layer is 1 as the target column (IS_SUCESSFUL) is binary
output_layer_size = 1
print("Output Layer Size:",output_layer_size)

# The number of hidden neurons here is less than twice the size of the input layer
hidden_nodes_1 = (input_features * 2) - 1
print("Hidden nodes in layer 1:", hidden_nodes_1)

# The number of hidden neurons here is 2/3 the size of the input layer, plus the size of the output layer
nodes2 = input_features * (2/3)
hidden_nodes_2 = round(nodes2,0) + output_layer_size
print("Hidden nodes in layer 2:", int(hidden_nodes_2))
```

```
Input Features: 43
Output Layer Size: 1
Hidden nodes in layer 1: 85
Hidden nodes in layer 2: 30
```

- **Result**: The model based on the parameters above gave an accuracy of 73.11%, which did not reach the target of 75%.

```
# 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: 0.5650 - accuracy: 0.7312 - 559ms/epoch - 2ms/step
Loss: 0.5650197267532349, Accuracy: 0.731195330619812
```

The model was further optimized multiple times to see if an accuracy of 75% could be attained.

Optimization 1

The following changes were done to the model as part of the first optimization attempt:

- The APPLICATION_TYPEs with a count of less than 100 (reduced from 500 in the model above) were binned as “Other”.
- The CLASSIFICATIONs with a count of less than 100 (reduced from 1000 in the model above) were binned as “Other”.
- One more hidden layer was added to the existing layers.
- The number of input features was increased to 50 (from 43).
- The hidden nodes were 99, 99 and 34 respectively on layers 1, 2 and 3.

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.

# Size of data is selected as the number of input features here
input_features = X_train_scaled.shape[1]
print("Input Features:",input_features)

# Size of the output layer is 1 as the target column (IS_SUCESSFUL) is binary
output_layer_size = 1
print("Output Layer Size:",output_layer_size)

# The number of hidden neurons here is less than twice the size of the input layer
hidden_nodes_1 = (input_features * 2) - 1
print("Hidden nodes in layer 1:", hidden_nodes_1)

# The number of hidden neurons here is less than twice the size of the input layer
hidden_nodes_2 = (input_features * 2) - 1
print("Hidden nodes in layer 2:", hidden_nodes_2)

# The number of hidden neurons here is 2/3 the size of the input layer, plus the size of the output layer
nodes3 = input_features * (2/3)
hidden_nodes_3 = round(nodes3,0) + output_layer_size
print("Hidden nodes in layer 3:", int(hidden_nodes_3))
```

```
Input Features: 50
Output Layer Size: 1
Hidden nodes in layer 1: 99
Hidden nodes in layer 2: 99
Hidden nodes in layer 3: 34
```

- **Result**: The model based on the parameters above gave an accuracy of 73.01%, which did not reach the target of 75%.

```
# 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: 0.6132 - accuracy: 0.7301 - 594ms/epoch - 2ms/step
Loss: 0.6131804585456848, Accuracy: 0.7301457524299622
```

Optimization 2

- The CLASSIFICATIONs with a count of less than 150 (increased from 100 in the model above) were binned as “Other”.
- The number of input features was reduced to 47 (from 50).
- The hidden nodes were 93, 93 and 32 respectively on layers 1, 2 and 3.

```
Input Features: 47
Output Layer Size: 1
Hidden nodes in layer 1: 93
Hidden nodes in layer 2: 93
Hidden nodes in layer 3: 32
```

- The activation function “tanh” used on layers 1 and 2 (“relu” in the previous model).

```
# Create a Keras Sequential model
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_1, activation="tanh", input_dim=input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_2, activation="tanh"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_3, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units=output_layer_size, activation="sigmoid"))

# Check the structure of the model
nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 93)                4464

 dense_1 (Dense)             (None, 93)                8742

 dense_2 (Dense)             (None, 32)                3008

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 16,247
Trainable params: 16,247
Non-trainable params: 0
_________________________________________________________________
```

- **Result**: The model based on the parameters above gave an accuracy of 72.85%, which is less than the previous models.

```
# 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: 0.5791 - accuracy: 0.7285 - 628ms/epoch - 2ms/step
Loss: 0.5790967345237732, Accuracy: 0.7285131216049194
```

Optimization 3

- A function to create a new Sequential model with hyperparameter options was used.
- The function used kerastuner to automate the tuning of the hyperparameters and select the best of the activation functions, number of neurons in each layer and the number of hidden layers.
- The epochs was reduced to 20 (100 in the previous models).
- The best accuracy value obtained by the automation is 73.82%.

```
# Run the kerastuner search for best hyperparameters

tuner.search(X_train_scaled,y_train,epochs=20,validation_data=(X_train_scaled,y_train))
```

```
Trial 60 Complete [00h 01m 12s]
val_accuracy: 0.7374436259269714

Best val_accuracy So Far: 0.7382988929748535
Total elapsed time: 00h 29m 55s
INFO:tensorflow:Oracle triggered exit
```

- The best model hyperparameters identified by the automation are as follows:

```
# Get best model hyperparameters
best_hyper = tuner.get_best_hyperparameters(1)[0]
best_hyper.values
```

```
{'activation': 'tanh',
 'first_units': 29,
 'num_layers': 6,
 'units_0': 33,
 'units_1': 27,
 'units_2': 13,
 'units_3': 17,
 'units_4': 29,
 'units_5': 7,
 'tuner/epochs': 20,
 'tuner/initial_epoch': 0,
 'tuner/bracket': 0,
 'tuner/round': 0}
```

x

- **Result**: The model based on the parameters above gave an accuracy of 73.44%, which is less than the previous models.

```
# Evaluate best model against full test data
best_model = tuner.get_best_models(1)[0]
model_loss, model_accuracy = best_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 1s - loss: 0.5515 - accuracy: 0.7345 - 982ms/epoch - 4ms/step
Loss: 0.5515479445457458, Accuracy: 0.7344606518745422
```

Optimization 4

- While the variable NAME is added back to the input data, the variables EIN, ORGANIZATION and SPECIAL CONSIDERATIONS are removed.
- The APPLICATION_TYPEs with a count of less than 500 (increased from 100 in the previous models) are binned as “Other”.
- The CLASSIFICATIONs with a count of less than 1000 (increased from 150 in the previous models) are binned as “Other”.
- The NAMEs with a count of less than 21 are binned as “Other”. This variable was dropped in the previous models.
- The number of layers reduced by 1 from the previous models.
- The input features increased to 156 (this is because of the NAME column added back to the input) thereby increasing the hidden nodes to 311 and 105 respectively on layers 1 and 2.

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.

# Size of data is selected as the number of input features here
input_features = X.shape[1]
print("Input Features:",input_features)

# Size of the output layer is 1 as the target column (IS_SUCESSFUL) is binary
output_layer_size = 1
print("Output Layer Size:",output_layer_size)

# The number of hidden neurons here is less than twice the size of the input layer
hidden_nodes_1 = (input_features * 2) - 1
print("Hidden nodes in layer 1:", hidden_nodes_1)

# The number of hidden neurons here is 2/3 the size of the input layer, plus the size of the output layer
nodes2 = input_features * (2/3)
hidden_nodes_2 = round(nodes2,0) + output_layer_size
print("Hidden nodes in layer 2:", int(hidden_nodes_2))
```

```
Input Features: 156
Output Layer Size: 1
Hidden nodes in layer 1: 311
Hidden nodes in layer 2: 105
```

- The activation functions used in both the layers is “tanh”.

- **Result**: The model based on the parameters above gave an accuracy of 76.73%, which has surpassed the requirement of 75%.

```
# 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}")
```

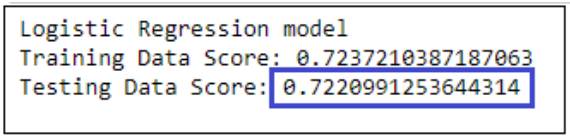
```
268/268 - 1s - loss: 0.4783 - accuracy: 0.7673 - 829ms/epoch - 3ms/step
Loss: 0.4782887101173401, Accuracy: 0.7673469185829163
```

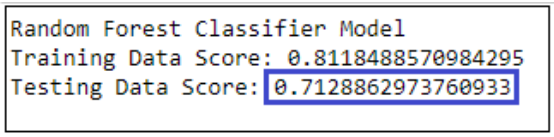
Results Summary

- The first model that was created by removing EIN and NAME gave an accuracy score of 73.11 which is less than the target of 75%.
- The model was then optimized multiple times by doing the following:
    i. adding a hidden layer
    ii. increasing/ decreasing the input features, hidden neurons and epochs
    iii. changing the way of binning the values of the columns APPLICATION TYPE and CLASSIFICATION
    iv. using different activation functions
    v. automating the tuning of the hyperparameters to select the best of the activation functions, number of layers and hidden nodes
    vi. adding and removing different sets of columns from the input datasets
- Finally, only after adding the NAME column back to the input did the accuracy score hit the target of 75%. The accuracy is now 76.73%. This may not be the correct way of doing as the name of an organization should not decide the funding but the use case and other factors.
- There was not much of a difference in the accuracy scores generated by all models above, using both the training and test data.

Recommendation

- As the target values (IS_SUCCESSFUL) are available already, Supervised Machine Learning techniques can be considered as well, for building the tool.
- Training using the supervised machine language models Logistic Regression and Random Forest Classifier did not give the expected results. Both models gave an accuracy score of around 72%.
- However, Random Forest Classifier model gave a training score of 81%. So, this can be looked at in depth in comparison to the neural networks which costs high.





References

Krishnan, S. (2021, September 8). *Medium*. Retrieved from Medium: https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3