Module 21 Deep Learning Challenge – Charity Funding Predictor

Overview

- The purpose of this analysis was to develop a tool the Alphabet Soup can use to help select applicants for funding that would have the best chance of success in their ventures.
- This report outlines how the model was created and the results from it. It is divided into two sections:
  - Section One – Initial model
  - Section two – model optimisation.

Section 1: Initial Model

Data pre-processing

- The target variable for the model was whether the funding money had been used effectively, identified via the column 'IS_SUCCEESSFUL'. With '1' being successful and '0' not.
- The feature variables of the model are listed below. As per the instructions the identification columns 'EIN' and 'NAME' were removed.

APPLICATION_TYPE	Alphabet Soup application type
AFFILIATION	Affiliated sector of industry
CLASSIFICATION	Government organisation classification
USE_CASE	Use case for funding
ORGANIZATION	Organisation type
STATUS	Active status
INCOME_AMT	Income classification
SPECIAL_CONSIDERATIONS	Special considerations for application
ASK_AMT	Funding amount requested

- For the field APPLICATION_TYPE, any with a value count less than 500 were considered small enough to be grouped together under the category 'other'.
- For the field CLASSIFICTION any with a value count less than 1,000 were considered small enough to be grouped tougher under the category 'other'.
- The categorical data was then converted to numeric.
- The dependent variable ('IS_SUCCESSFUL) was removed from the independent variables.
- The data was split into training and testing datasets.
- After which StandardScaler was applied to the data to resize the distribution of values.

Compile, train and evaluate the model:

- This model was built with the aim of obtaining an initial understanding of how the Neural network performed prior to optimisation, with the parameters used listed below.
- Given the dataset was not overly complex two hidden layers were used for the initial model.
- 100 epochs were used as it was thought to be a good starting point for the initial model.

```
[17] # 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 =  30
     hidden_nodes_layer2 = 20

     nn_model = tf.keras.models.Sequential()

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

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

     # Output layer
     nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

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

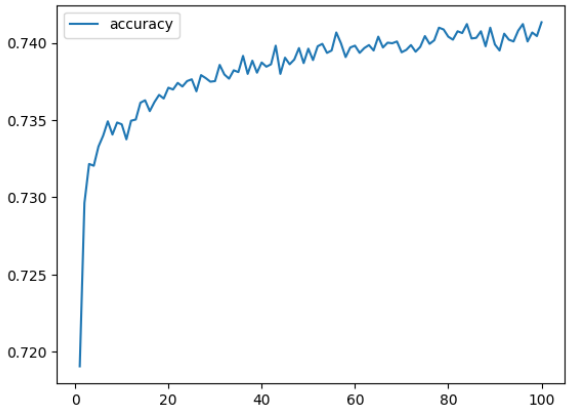
```
[20] # Train the model
     fit_model = nn_model.fit(X_train_scaled, y_train, epochs=100, callbacks=[callback])
```

	Number of neurons used	Activation function
Input layer	Set to use all inputted data	The ReLU activation function was used as it is a simple function that has been proven to work very well in deep neural networks.
First hidden layer	Set to 30 – based on it being between the size of the input features and output layer	
Second hidden layer	Set as 20 - based on it being between the size of the input features and output layer	
Output layer	Set to be 1 as only need to know whether it was successful	Sigmoid activation used as it is commonly used for models where the model output is a binary classification

Model evaluation

- The model has an accuracy rate of 0.7245, below the target accuracy of 0.75. It also has a loss of 0.5596

```
268/268 - 0s - loss: 0.5596 - accuracy: 0.7245 - 305ms/epoch - 1ms/step
Loss: 0.5595604777336121, Accuracy: 0.7245481014251709
```



Section 2: Model Optimisation

Data pre-processing

- Due to the low accuracy of the first model, the inputs were reviewed. As a result, the 'NAME' column was included. The reason for including it was because some applicates had received funding hundreds of times, which may have some impact on success.
- For the field NAME, any with a value count of less than 100 were considered small enough to be grouped together under the category 'other'.
- Given this change the feature variables used for the development of the model were:

NAME	Name of the group/organisation receiving funding
APPLICATION_TYPE	Alphabet Soup application type
AFFILIATION	Affiliated sector of industry
CLASSIFICATION	Government organisation classification
USE_CASE	Use case for funding
ORGANIZATION	Organisation type
STATUS	Active status
INCOME_AMT	Income classification
SPECIAL_CONSIDERATIONS	Special considerations for application
ASK_AMT	Funding amount requested

Attempt 1 – compile, train and evaluate the model

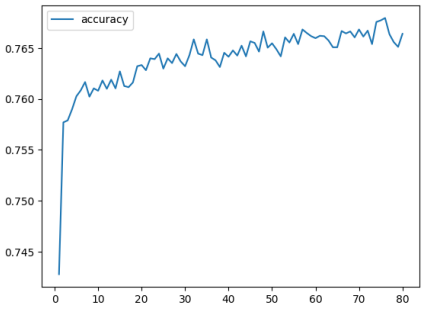
- The accuracy chart from the initial model indicated accuracy plateaued at around 80 epochs. Based on this the number of epochs were reduced from 100 to 80.
- Everything else remained the same to help understand the impact of including 'NAME' in the input data.

```
# 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 =  30
hidden_nodes_layer2 = 20

nn_model_opt_1 = tf.keras.models.Sequential()

# First hidden layer
nn_model_opt_1.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn_model_opt_1.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer
nn_model_opt_1.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn_model_opt_1.summary()
```

```
# Train the model
fit_model = nn_model_opt_1.fit(X_train_scaled, y_train, epochs=80, callbacks=[callback])
```

- These changes improved the accuracy, with it achieving a score of 0.7524, slightly above the accuracy goal of 0.75.
-

```
268/268 - 0s - loss: 0.5004 - accuracy: 0.7524 - 489ms/epoch - 2ms/step
Loss: 0.5004160404205322, Accuracy: 0.7524198293685913
```



Attempt 2 – compile, train and evaluate the model

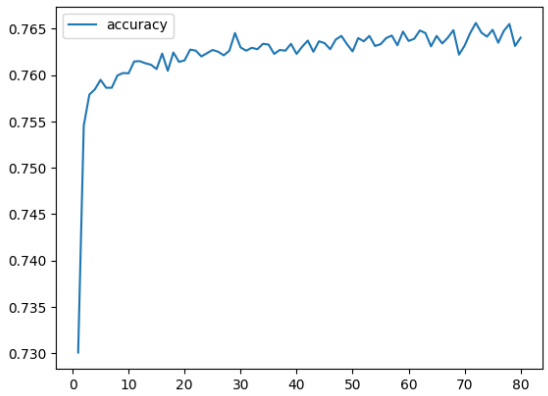
- To try and further improve the accuracy the second hidden layer was removed, to see if simplifying the model would help. The reason for this change was several articles noting one hidden layer can be sufficient for less complicated models[1]

```
# 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 =  30

nn_model_opt_2 = tf.keras.models.Sequential()

# First hidden layer
nn_model_opt_2.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu"))

# Output layer
nn_model_opt_2.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn_model_opt_2.summary()
```

```
# Train the model
fit_model = nn_model_opt_2.fit(X_train_scaled, y_train, epochs=80, callbacks=[callback])
```

- This change reduced the accuracy of the model to 0.7497, below the accuracy threshold of 0.75

```
268/268 - 0s - loss: 0.4948 - accuracy: 0.7497 - 475ms/epoch - 2ms/step
Loss: 0.4948359429836273, Accuracy: 0.7497376203536987
```



[1] https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3

Attempt 3 – compile, train and evaluate the model

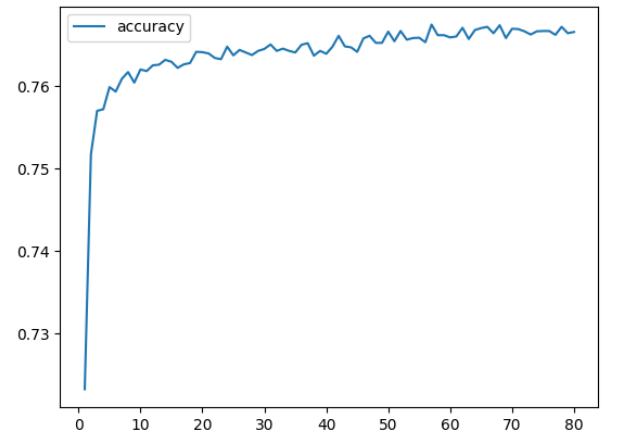
- Given removing a hidden layer reduced the accuracy of the model, in the last attempt to optimise the model both hidden layers were reinstated, and an additional layer added with 5 neurons. The reason for 5 neurons was to continue reducing the number of neurons in the model.

```
# 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 =  40
hidden_nodes_layer2 =  10
hidden_nodes_layer3 =  5

nn_model_opt_3 = tf.keras.models.Sequential()

# First hidden layer
nn_model_opt_3.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn_model_opt_3.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Third hidden layer
nn_model_opt_3.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation="relu"))

# Output layer
nn_model_opt_3.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn_model_opt_3.summary()
```

```
# Train the model
fit_model = nn_model_opt_3.fit(X_train_scaled, y_train, epochs=80, callbacks=[callback])
```

- This change resulted in the highest accuracy score achieved with this model at 0.7531

```
268/268 - 1s - loss: 0.4954 - accuracy: 0.7531 - 658ms/epoch - 2ms/step
Loss: 0.4954368770122528, Accuracy: 0.7531195282936096
```



Conclusion

The model prior to optimisation had an accuracy score of 0.7245. Three attempts were made to optimise it, with the final attempt producing the highest accuracy score of 0.7531.

To optimise the model, the following aspects were undertaken:

1. Inclusion of the 'NAME' column
    - Due to the low accuracy of the first model, the inputs were reviewed. As a result, the 'NAME' column was included. The reason for including it was because some applicates had received funding hundreds of times, which may have some impact on success.
2. Changed epochs from 100 to 80
    - The accuracy chart from the initial model indicated that accuracy plateaued at around 80 epochs. Based off this the number of epochs were changed to 80 for the optimisation attempts.
3. Changed the number of hidden layers
    - Initially the model had two hidden layers, in an attempt to optimise the model one was removed. The reason for this was based off several articles [2]noting one hidden layer is often sufficient for simple datasets. However, this resulted in a decline in accuracy for this model.
    - As a result, a third hidden layer was included, which provided the highest accuracy of all models at 0.7531.

Alternative model

I would recommend Random Forest as an alternative to Neural networks in this instance. Largely due to the ease of use and simplicity of Random Forest, but also because out of all the classification methods, Random Forest performs very strongly in terms of accuracy.

There are several benefits of using a Random Forest instead of a Neural network:

- Performance – Random Forests are less taxing on the computer to run, can be faster and require less training time than Neural networks, particularly on smaller datasets with less features.
- Interpretability – Random forests are more interpretable than Neural networks as they provide importance scores that allow you to understand which features are most important for the model. Neural networks on the other hand are very much a black box, making it difficult to explain the inner workings of the model to others.
- Robustness – Random forests are robust to outliers and noise in the data, while Neural networks can be quite sensitive to this.

[2] https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3