Analysis: Deep Learning model for AlphabetSoup data

Overview

Analysis of the performance of deep learning model created for AlphabetSoup. This report describes the process for creating and optimising a deep learning neural network to predict whether a charity will successfully use funds granted to them.

Data Pre-processing

The target for my model is the column titled 'Is_successful'.

All variables, aside from the name and id columns, are features for the model. The columns name and id should be removed as they are neither targets nor features.

Compiling, Training, and Evaluating the Model

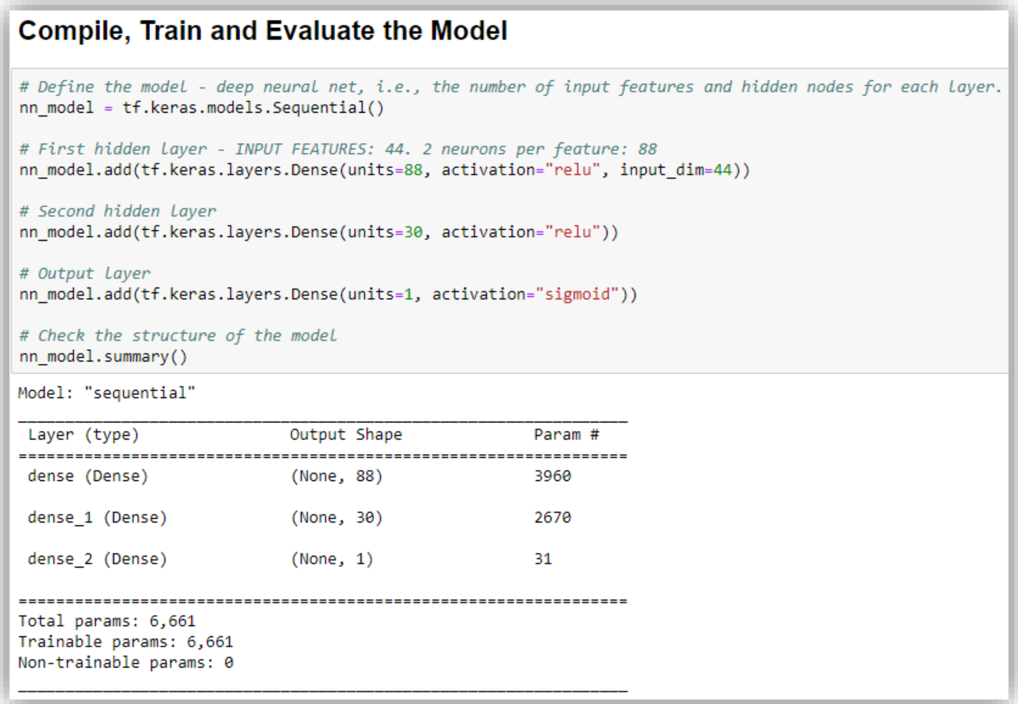
I initially wanted to use keras-tuner to find the best parameters for optimising the model, however the size of the dataset made that option unfeasible.

My approach was to use best practice recommendations for hyperparameters, starting with simple models and then increasing complexity.

Best practice:

- 2-4 hidden layers
- 2-3 neurons per feature
- Sigmoid activation function for the outer layer and more complex activation functions (relu, tanh) for the hidden layers
- Minimum 100 Epochs

My first model had two layers, 2 neurons per input dimension (88 neurons in the first layer), 100 Epochs. The accuracy score was 0.7298.



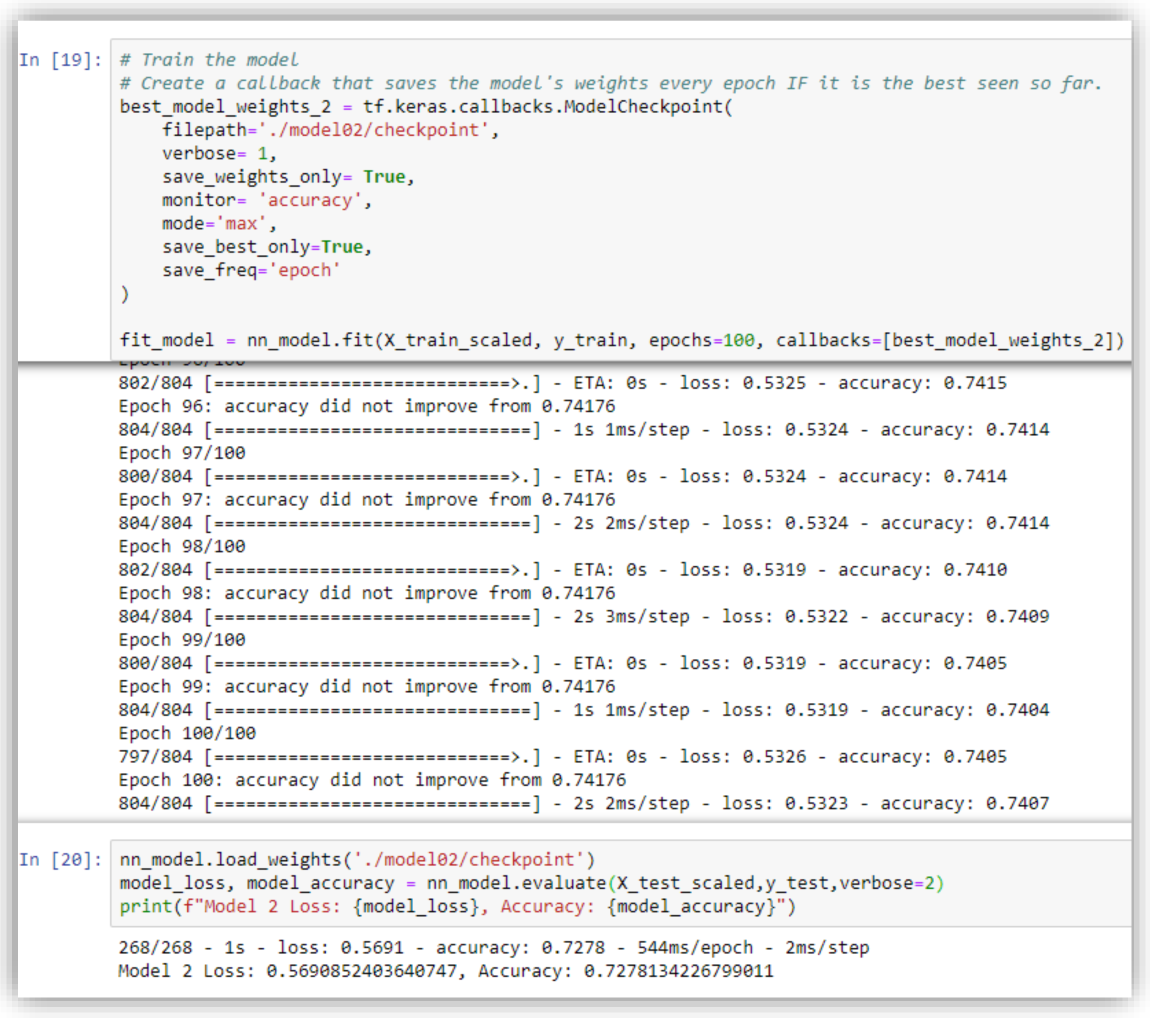
Compile, Train and Evaluate the Model

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
nn_model = tf.keras.models.Sequential()

# First hidden layer - INPUT FEATURES: 44. 2 neurons per feature: 88
nn_model.add(tf.keras.layers.Dense(units=88, activation="relu", input_dim=44))

# Second hidden layer
nn_model.add(tf.keras.layers.Dense(units=30, activation="relu"))

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 88)                3960

 dense_1 (Dense)             (None, 30)                2670

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 6,661
Trainable params: 6,661
Non-trainable params: 0
_________________________________________________________________
```

Figure 1. Model 1.

<u>Model 2:</u> I loaded the same model with the best weights for accuracy to see how this function works. There was no improvement in performance.



```
In [19]: # Train the model
         # Create a callback that saves the model's weights every epoch IF it is the best seen so far.
         best_model_weights_2 = tf.keras.callbacks.ModelCheckpoint(
             filepath='./model02/checkpoint',
             verbose= 1,
             save_weights_only= True,
             monitor= 'accuracy',
             mode='max',
             save_best_only=True,
             save_freq='epoch'
         )

         fit_model = nn_model.fit(X_train_scaled, y_train, epochs=100, callbacks=[best_model_weights_2])
```

```
802/804 [============================>.] - ETA: 0s - loss: 0.5325 - accuracy: 0.7415
Epoch 96: accuracy did not improve from 0.74176
804/804 [==============================] - 1s 1ms/step - loss: 0.5324 - accuracy: 0.7414
Epoch 97/100
800/804 [============================>.] - ETA: 0s - loss: 0.5324 - accuracy: 0.7414
Epoch 97: accuracy did not improve from 0.74176
804/804 [==============================] - 2s 2ms/step - loss: 0.5324 - accuracy: 0.7414
Epoch 98/100
802/804 [============================>.] - ETA: 0s - loss: 0.5319 - accuracy: 0.7410
Epoch 98: accuracy did not improve from 0.74176
804/804 [==============================] - 2s 3ms/step - loss: 0.5322 - accuracy: 0.7409
Epoch 99/100
800/804 [============================>.] - ETA: 0s - loss: 0.5319 - accuracy: 0.7405
Epoch 99: accuracy did not improve from 0.74176
804/804 [==============================] - 1s 1ms/step - loss: 0.5319 - accuracy: 0.7404
Epoch 100/100
797/804 [============================>.] - ETA: 0s - loss: 0.5326 - accuracy: 0.7405
Epoch 100: accuracy did not improve from 0.74176
804/804 [==============================] - 2s 2ms/step - loss: 0.5323 - accuracy: 0.7407
```

```
In [20]: nn_model.load_weights('./model02/checkpoint')
         model_loss, model_accuracy = nn_model.evaluate(X_test_scaled,y_test,verbose=2)
         print(f"Model 2 Loss: {model_loss}, Accuracy: {model_accuracy}")

         268/268 - 1s - loss: 0.5691 - accuracy: 0.7278 - 544ms/epoch - 2ms/step
         Model 2 Loss: 0.5690852403640747, Accuracy: 0.7278134226799011
```

Figure 2. Model 2 callback function and results.

<u>Model 3:</u> Add third hidden layer.

<u>Model 4:</u> Add fourth hidden layer, increase epochs to 200.

<u>Model 5:</u> Add fifth hidden layer, increase epochs to 200, include sigmoid, relu and tanh activation functions.

There was no improvement with any of these models and I was not able to achieve the target performance.

Further Pre-processing

My next attempt at improving the model was to adjust the input data, then test each of the five models with the new input data.

I viewed the value counts for each column to determine if there were any rare occurrences. I found that the STATUS column and the SPECIAL_CONSIDERATIONS column both had a gross imbalance of values. I removed the rows that were rare occurrences, then removed the columns altogether.

The result was a data frame with 42 columns, 41 input features and 32 less rows.

Compile, Train and Evaluate Models using new input.

I trained and tested each model on the new dataset, but the outcome was the same as before. There was no improvement in model performance and the accuracy score remained at 72-73%.

Summary

The deep learning model did not achieve target accuracy of >75%. A random forests or logistical regression model may be able to solve this problem as they have been effective for a similar case (predicting loan approval). Random forests may struggle with the size of this dataset.

Perhaps - with either greater computing capacity or a smaller dataset - keras-tuner could be used to find ideal parameters and increase model performance.