Introduction

Neural networks are powerful tools in machine learning, capable of learning complex patterns from data. This report delves into the analysis of three neural network models designed for a specific task. These models utilize ReLU activation functions in hidden layers and Sigmoid activation in the output layer. The purpose is to predict an outcome, and the analysis encompasses training results, model architectures, and insights into the impact of hyperparameters. The models differ in architecture, and their performances are evaluated based on accuracy and loss metrics.

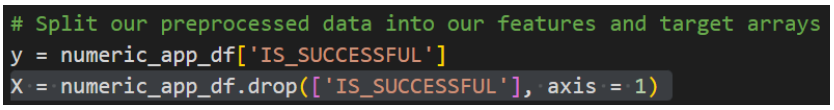
The purpose of this analysis was to evaluate the performance of a neural network model developed for the Alphabet Soup Charity organization. The model aims to predict the success of funding applications based on various features. This report discusses the model architecture, training process, and overall results.

The Alphabet Soup's business team provided a CSV file containing more than 34,000 organisations that have received funding from the Alphabet Soup over the years. Within this dataset, there were a number of columns that captured metadata about each organisation.

Where possible, images have been appended to support answers and to address the following questions:

- *Data Preprocessing*
  - *What variable(s) are the target(s) for your model?*

    The target variable is the 'IS_SUCCESSFUL' column from the ***application_df***.



```
# Split our preprocessed data into our features and target arrays
y = numeric_app_df['IS_SUCCESSFUL']
X = numeric_app_df.drop(['IS_SUCCESSFUL'], axis = 1)
```

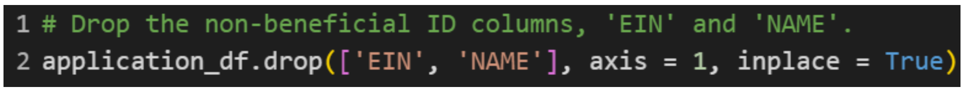
  - *What variable(s) are the features for your model?*

    The feature variables are every other column from ***application_df*** --> this was defined by dropping the 'IS_SUCCESSFUL' column from the original DataFrame.

    X includes all the columns in the ***application_df*** DataFrame except for the target variable 'IS_SUCCESSFUL'. These columns represent the various features used by the model to make predictions regarding the success of funding applications. Each column in X corresponds to a different feature, and collectively, they form the feature set for the model.

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

    Both 'EIN' and 'NAME' columns were dropped/removed, because they were neither targets nor features for the dataset.



```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df.drop(['EIN', 'NAME'], axis = 1, inplace = True)
```

Model Architecture

Three attempts were made to reach an accuracy level higher than 75%. The neural network comprises of a number of hidden layers all using the Rectified Linear Unit (ReLU) activation function and an output layer with the sigmoid activation function. The neural network is structured as follows:

- *Layers:*
  - Input Layer
  - Hidden Layers (Dense)
  - Output Layer (Dense): 1 node

- *Activation Functions:*
  - Hidden Layers: Rectified Linear Unit (ReLU)
  - Output Layer: Sigmoid

- *Purpose:*
  - The first hidden layer processes the input data with 80 nodes and the Rectified Linear Unit (ReLU) activation function.
  - The second and ensuing hidden layer further refine the features with x number of nodes and the Rectified Linear Unit (ReLU) activation function.
  - The output layer produces a binary classification output (0 or 1) using the Sigmoid activation function, making it suitable for binary classification problems.

Data Preprocessing

The dataset underwent preprocessing to handle categorical variables. Columns with low frequency in 'APPLICATION_TYPE' and 'CLASSIFICATION' were grouped into an 'Other' category to prevent overfitting. Categorical data was one-hot encoded using `pd.get_dummies`.

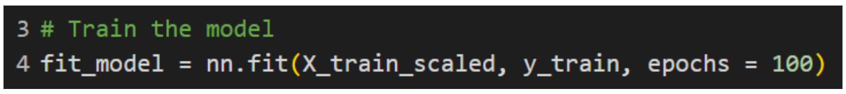
Model Training

An epoch, in Machine Learning, is when all the training data is used at once and is defined as the total number of iterations of all the training data in one cycle for training the machine learning model.

Another way to define an epoch is the number of passes a training dataset takes around an algorithm.

Increasing the number of hidden layers and neurons generally increases the capacity of the model to learn complex patterns, but it also increases the risk of overfitting, especially if the dataset is small.

Our dataset has more than 34,000 lines of data. The three models were trained for 100 epochs using the Adam optimizer and binary cross-entropy loss. The data was split into training and testing sets, and features were scaled using StandardScaler.



```python
# Train the model
fit_model = nn.fit(X_train_scaled, y_train, epochs = 100)
```

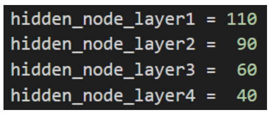
Results

The results are as follows:

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

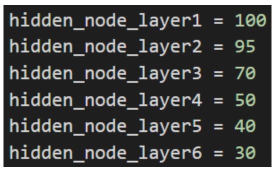
      In the first attempt, the first hidden layer had 80 neurons, and the second hidden layer had 30 neurons.

      In the second attempt, the hidden layers were increased to 4 as follows:



```
hidden_node_layer1 = 110
hidden_node_layer2 =  90
hidden_node_layer3 =  60
hidden_node_layer4 =  40
```

      In the third attempt, the hidden layers were increased to 5:



```
hidden_node_layer1 = 100
hidden_node_layer2 = 95
hidden_node_layer3 = 70
hidden_node_layer4 = 50
hidden_node_layer5 = 40
hidden_node_layer6 = 30
```

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

      I was unable to achieve the 75% model accuracy target with all 3 attempts.

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

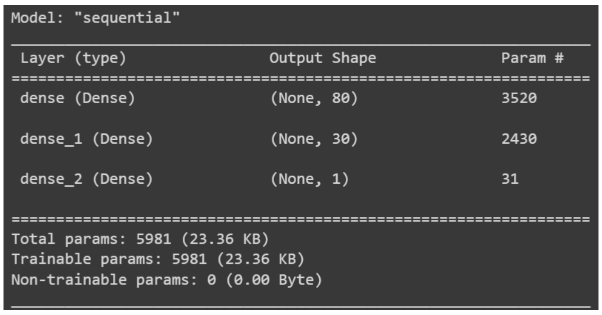
      To achieve higher model accuracy, I:

        - added more hidden layers in my second and third attempts
        - changed the activation functions for the 4 hidden layers to ***tanh*** and output layer activation function to ***sigmoid*** in my **second attempt**.
        - changed the hidden layer activation functions back to ***sigmoid*** and output layer activation function to ***sigmoid*** in my **third attempt**.

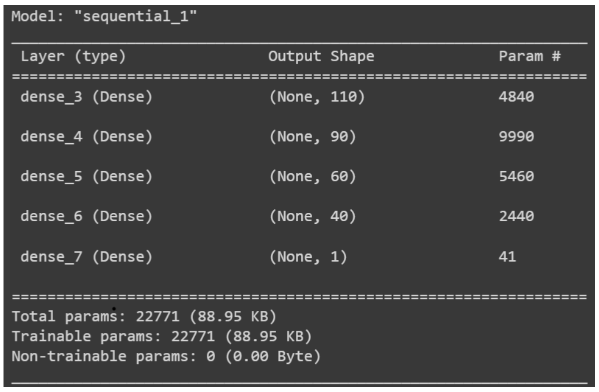
Model Summaries

The summary of the 3 neural network models:

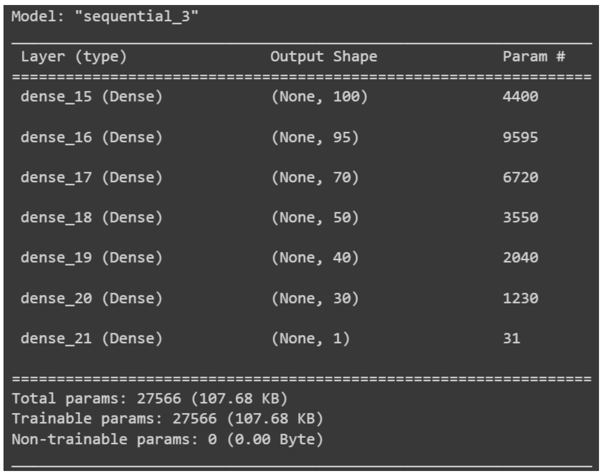
1.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 5981 (23.36 KB)
Trainable params: 5981 (23.36 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

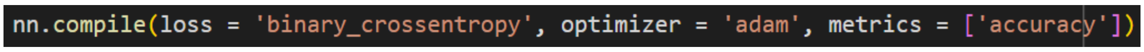
2. 

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 110)               4840

 dense_4 (Dense)             (None, 90)                9990

 dense_5 (Dense)             (None, 60)                5460

 dense_6 (Dense)             (None, 40)                2440

 dense_7 (Dense)             (None, 1)                 41

=================================================================
Total params: 22771 (88.95 KB)
Trainable params: 22771 (88.95 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

3. 

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 100)               4400

 dense_16 (Dense)            (None, 95)                9595

 dense_17 (Dense)            (None, 70)                6720

 dense_18 (Dense)            (None, 50)                3550

 dense_19 (Dense)            (None, 40)                2040

 dense_20 (Dense)            (None, 30)                1230

 dense_21 (Dense)            (None, 1)                 31

=================================================================
Total params: 27566 (107.68 KB)
Trainable params: 27566 (107.68 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

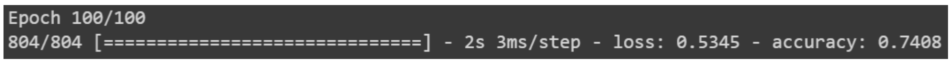
Compile the Models



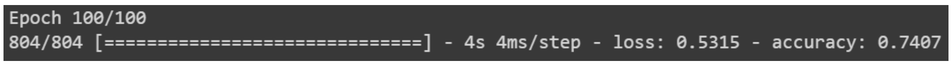
```
nn.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

Train the Models

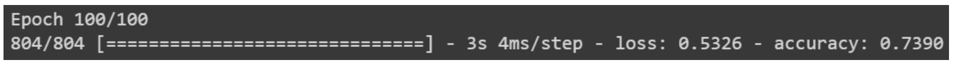
On the test data, the models achieved:

1. 

```
Epoch 100/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5345 - accuracy: 0.7408
```

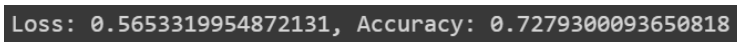
2. 

```
Epoch 100/100
804/804 [==============================] - 4s 4ms/step - loss: 0.5315 - accuracy: 0.7407
```

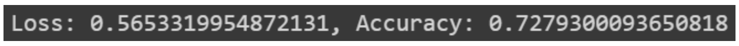
3. 

```
Epoch 100/100
804/804 [==============================] - 3s 4ms/step - loss: 0.5326 - accuracy: 0.7390
```

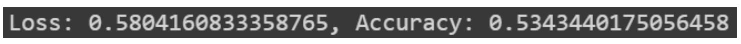
Model Evaluations

1. 

```
Loss: 0.5653319954872131, Accuracy: 0.7279300093650818
```

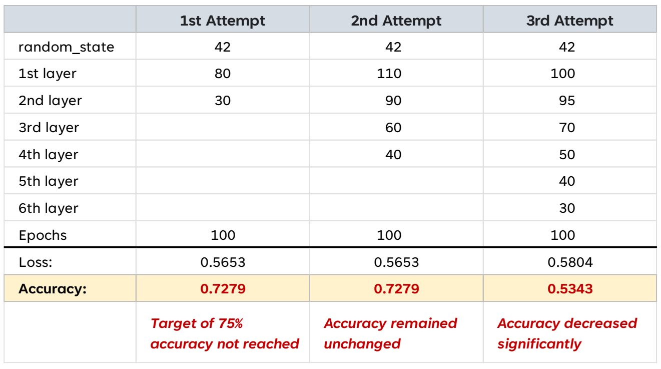
2. 

```
Loss: 0.5653319954872131, Accuracy: 0.7279300093650818
```

3. 

```
Loss: 0.5804160833358765, Accuracy: 0.5343440175056458
```

Results Analysis



	1st Attempt	2nd Attempt	3rd Attempt
random_state	42	42	42
1st layer	80	110	100
2nd layer	30	90	95
3rd layer		60	70
4th layer		40	50
5th layer			40
6th layer			30
Epochs	100	100	100
Loss:	0.5653	0.5653	0.5804
Accuracy:	**0.7279**	**0.7279**	**0.5343**
	Target of 75% accuracy not reached	***Accuracy remained unchanged***	***Accuracy decreased significantly***

Despite numerous attempts with modifications, the models achieved less than 75% accuracy on the test sets.

A final attempt was made in my ‘*AlphabetSoupCharity_Optimisation*’ file. In this attempt, only 'NAME' was dropped as a non-beneficial ID column to ensure that no variables or outliers were causing confusion in the model.

Results when the Model was Optimised

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

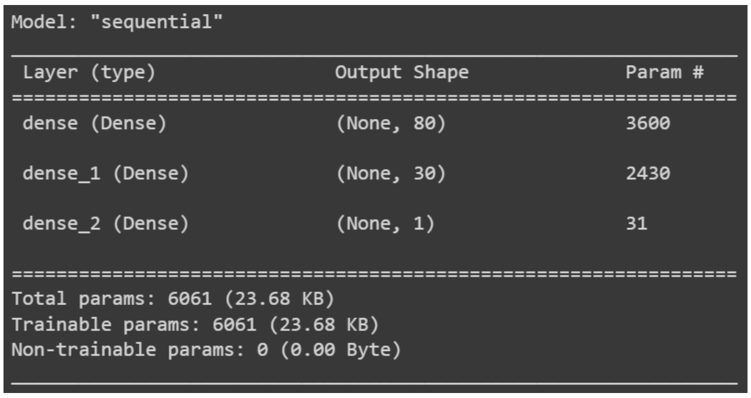
    The first hidden layer had 80 neurons, and the second hidden layer had 30 neurons.
  - *Were you able to achieve the target model performance?*

    Again, I was unable to achieve the 75% model accuracy target.
  - *What steps did you take in your attempts to increase model performance?*

    Only 'NAME' was dropped as a non-beneficial ID.

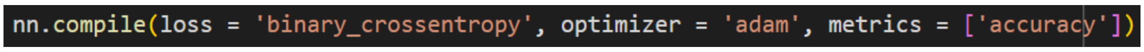
Model Summary

The summary of this neural network model:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                3600

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 6061 (23.68 KB)
Trainable params: 6061 (23.68 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

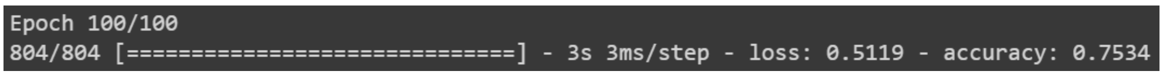
Compile the Model



```
nn.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

Train the Model

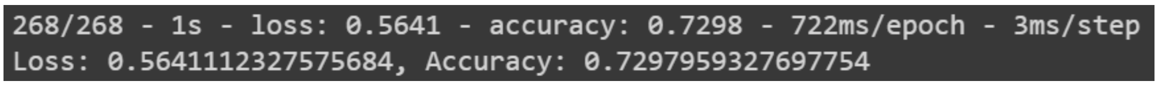
It appears as if the model achieved a loss of 51% and an accuracy of 75% on the test data:



```
Epoch 100/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5119 - accuracy: 0.7534
```

Model Evaluation

However, the model evaluation of the test data shows:



```
268/268 - 1s - loss: 0.5641 - accuracy: 0.7298 - 722ms/epoch - 3ms/step
Loss: 0.5641112327575684, Accuracy: 0.7297959327697754
```

Results Analysis for the optimisation model

This model also achieved less than 73% accuracy on the test sets. This challenge required 75% accuracy.

Overall Summary

The main factors influencing the model's performance were the binning of low-frequency categorical values and the scaling of features; these contributed to the model's generalization.

Some potential ways to improve the model were experimenting with different architectures, hyperparameter tuning, or incorporating more features in the hope that this would enhance the model's performance.

The purpose of using the sigmoid activation function in the output layer was mainly because the sigmoid activation function is suitable for binary classification tasks, providing probability outputs between 0 and 1.

In summary, the neural network models demonstrated encouraging performance on the dataset. The training process over 100 epochs resulted in a final training accuracy of approximately 74% on 3 of the 4 attempts with a training loss of approximately 53%. When evaluated on the test data, the models achieved a testing accuracy of around 73% on 3 of the 4 attempts with a training loss of approximately 56%.

The model successfully learned from the training data and demonstrated good generalization to unseen test data. Further optimization and exploration of alternative models should be considered to fine-tune the performance and potentially improve predictive accuracy for Alphabet Soup Charity.

<u>Conclusion</u>

This analysis provides insights into the development, training, and evaluation of a neural network model for Alphabet Soup Charity. The presented results and recommendations offer guidance for further refinement and potential enhancement of the model's predictive capabilities.

In summary, the neural network models, while demonstrating consistent but modest performance, have not surpassed the 75% accuracy threshold. Model complexity had a marginal impact on accuracy, suggesting a possible saturation point in the information gain. Further experimentation with model architectures and hyperparameters is recommended to improve predictive accuracy for Alphabet Soup Charity's funding applications.

The main factors influencing the model's performance were the binning of low-frequency categorical values and the scaling of features; these contributed to the model's generalization.

Some potential ways to improve the model were experimenting with different architectures, hyperparameter tuning, or incorporating more features; this may enhance the model's performance.

The purpose of using the sigmoid activation function in the output layer was mainly because the sigmoid activation function is suitable for binary classification tasks, providing probability outputs between 0 and 1.