**CHALLENGE 21**

**Introduction**

This study is regarding creating a model to predict the outcome of ‘successful’ use of funding provided by the nonprofit foundation **Alphabet Soup**.

In other words, the conditions and parameters of requests made by applicants are assessed to predict if the goal of funding could be achieved by an applicant. This process would be helpful in predicting whether an applicant would be successful if funded by Alphabet Soup.

A **sequential neural network model** is used for the study. In this model:

1. **Neurons** or nodes are the basis of the model. Neurons receive input signals, process and transform the signals and create output signals, much like the neurons in a human brain.
2. Neurons may exist in multiple **layers**, such as:

* **Input layer** which receives the initial signal,
* **Hidden layers**, with each layer comprising of a set of neurons, adding to the complexity of signal processing,
* **Output layer**, which produces the transformed data

1. Non-linear Activation functions are used for processing such as:

* ‘tanh’,
* ‘ReLU’
* ‘sigmoid’.

1. **Loss function** is a measure of error between predicted values and actual values.
2. **Accuracy function** is the percentage of correctly classified instances with respect to the input target values.
3. **Adam** is an algorithm to adjust and readjust weights associated with each neuron during the optimization process

**Data**

Data file **charity\_data.csv** is used for the study.

Each application has the following attributes:

* **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

**Analysis**

**Part I**

Data is cleaned to simplify and to reduce the number of categorical data.

Identification columns, not useful in analysis, **EIN** and **NAME**, are dropped.

IS\_SUCCESSFUL feature is used as the **target**.

APPLICATION\_TYPE, AFFILIATION, CLASSIFICATION, USE\_CASE, ORGANIZATION, STATUS, INCOME\_AMT, SPECIAL\_CONSIDERATIONS, ASK\_AMT are the **features** used for the model.

Neural network (NN) model with TensorFlow as back end is used in this study.

NN API called Keras, a part of TensorFlow is used as ‘tf.keras’.

Input file is cleaned to simplify and reduce the number of categorical data. Dummy values of 0 or 1 is assigned to each category, to ensure all the columns have numeric values.

**Model Architecture**

Data is split into training and testing sets. Data is then standardized.

The following architecture was used. The summary() method describes the **architecture** of the NN model.

A screenshot of a computer program

Description automatically generated

* **Model: “Sequential\_1”** indicates that it is a sequential model, *i.e.*, the layers are stacked sequentially from input to output.
* **Layer (type)**: The three layers, dense, dense\_1, dense\_2 are layers of type **Dense**, which means each layer is completely functional in the network.
* **Output Shape**: This shows the number of neurons in each layer.

For example, in (None, 8) , ‘None’ refers to the batch size, which is not fixed. ‘None’, and ‘8’ refer to the number of neurons in each dimension. The dimension indicating ‘None’ is populated automatically by the model, depending on the number of examples in the training ‘batch’.

This is a very useful feature of this model that allows it flexibility to adjust to the size of the training data.

* **Param #** shows the number of trainable parameters in each layer. There are 352 parameters in the first layer. These parameters are the weights and biases of the layer.
* **Total params** is the sum of all trainable parameters.
* **Trainable params** are weights and biases that change during training.
* **Non-trainable params** come from layers such as ‘BatchNormalization’ and remain fixed/unaffected during training. In the present case it is 0.
* **Epoch**: During model-training, Epoch is a complete iteration through the entire training set.

1. Forward Pass: the model makes predictions using one ‘batch’ of training data and compares it with the actual target data to calculate the ‘loss’.
2. Backward Pass: Derivatives of the ‘loss’ with respect to the parameters are computed.
3. Parameter Update: The model’s parameters are updated to minimize ‘loss’, using an optimizing algorithm, ‘Adam’ in our case.
4. Repeat: The above 3 steps are repeated for each batch of training data.

After one Epoch is completed, the results are displayed.

In our case, a callback function is created which records the model’s weights every 5 Epochs.

**Model Outcome**

The neural network model ‘nn’ is evaluated with the test dataset. The snippet below shows the last 2 Epochs, and the final results of ‘loss’ and ‘accuracy’.

A screenshot of a computer code

Description automatically generated

‘verbose’ parameter is set to 2, implies the values of ‘loss’ and ‘accuracy’ is displayed for each batch during the evaluation.

The difference between the last 2 ‘accuracy’ values and the final value may be attributed to **Overfitting**.

The model could learn to fit the training data too closely, which could include noise, or specific patterns such as **‘outliers’**. This can result in higher accuracy in the training data compared to that with the testing data.

The testing data may have slightly different characteristics compared to the training data, which could lead to lower ‘accuracy’ value.

In the present case, the accuracy value is **0.7321** or 73.21%.

**Part II**

In order to introduce more choices than in the previous model so as to improve the model’s performance, KerasTuner is used to optimize the hyperparameters of NN deep learning model.

The dataset is cleaned further. Additional columns, ‘STATUS’ and ‘SPECIAL\_CONSIDERATIONS’ dropped. Data rows with minority values ‘0’ or ‘Y’ are eliminated to prevent imbalance within dataset. The number of ‘APPLICATION\_TYPE’, and ‘CLASSIFICATION’ include a few **more** categories than in the previous analysis.

A method (function) is created to compile a new sequential model with hyperparameter options.

KerasTuner selects between ‘ReLU’ and ‘tanh’ activation functions for each hidden layer.

KerasTuner chooses between 1 to 30 neurons in the first layer.

KerasTuner choose between 1 to 5 hidden layers, and chooses between 1 to 30 neurons in each hidden layer.

KerasTuner is imported and ‘Hyperband’ tuner instance is created.

The objective of this process is to create an architecture with many choices so as to improve the ‘accuracy’ value.

The best option is expected to yield the highest value of ‘accuracy’ and the lowest value of ‘loss’.

With the present input parameters, the optimum choices were:

'activation': 'tanh', 'first\_layer': 11, 'num\_layers': 2, layer\_1': 16, 'layer\_2': 11.

The results using the testing data yielded:

A screenshot of a computer code

Description automatically generated

The results remained the same as in Part I, despite the wide variety of choices as input parameters.

**Part III**

Using the approach in Part I, to create a model, with inputs of our choosing (rather than an optimizer function), three models are created as an improvement upon the model in Part I.

Data is cleaned further, to include **fewer** categories.

Model 1

Activation function: ‘ReLU’

A screenshot of a computer program

Description automatically generated

Results 1

A screenshot of a computer code

Description automatically generated

Model 2

Activation function: ‘tanh’

A screenshot of a computer program

Description automatically generated

Results 2

A screenshot of a computer code

Description automatically generated

Model 3

Activation function: ‘ReLU’

A white text with black text

Description automatically generated with medium confidence

Results 3

A screenshot of a computer code

Description automatically generated

In all three models in Part III, the results showed only minor variations from those in Part I and Part II.

**Conclusion**

Table 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | First Layer  Neurons (N) | Hidden  Layers (L), Neurons(N) | Activation  Function | Loss | Accuracy |
| Part I | Model 1 | N: 8 | L: 1, N: 3 | ReLU | 0.5542 | 0.7321 |
| Part II | Model Best | N: 11 | L: 1, N: 16 | tanh | 0.5514 | 0.7308 |
| L: 2, N: 11 |
| Part III | Model 1 | N: 11 | L: 1, N: 21 | ReLU | 0.5611 | 0.7278 |
| L: 2, N: 16 |
| Model 2 | N: 11 | L: 1, N: 21 | tanh | 0.5604 | 0.7274 |
| L: 2, N: 16 |
| Model 3 | N: 6 | L: 1, N: 21 | ReLU | 0.5639 | 0.7271 |
| L: 2, N: 16 |
| L: 3, N: 11 |

Table 1 shows that both ‘loss’ and ‘accuracy’ values deteriorated slightly, despite the added complexity in the successive models.

The reason lies in the data itself. There is only **one feature** in the input data that has continuous numeric values: ‘ASK\_AMT’. The range of values span from $5,000 to $8,597,806,340!

A screenshot of a computer screen

Description automatically generated

Figure 1, Figure 2 show box plots for this feature without and with outliers, respectively.

25% of the loan requests are for amounts > $12,000;

75% of the requests are for amounts < $12,000, and are in the neighborhood of $5,000.

Figure 1

A white rectangular object with a black rectangle

Description automatically generated

Figure 2

A white background with red dots

Description automatically generated

25% is a significant portion of the data that cannot be deleted in the data-cleaning phase, as it may change the character of the training data.

Therefore, with **25%** of the data as **outliers**, the model may **overfit** to include outliers which are ¼ of the entire dataset!

**Thus, the model created will have limited success in achieving high accuracy values no matter the complexity of the model**.