# Neural Network Analysis Report

## Summary

The purpose of this analysis is to predict if applicants will be successful if provided with funds by a charity donor organisation called Alphabet Soup.

This is analysed using a Deep Learning Neural Network Model and further optimised by using optimisation options of the model. The output we expect is whether a given applicant will be successful or not, thus we use a binary classifier in this model.

The basic deep learning neural network provided an accuracy of ~72%. The optimised model also provided an accuracy of ~72%.

Perhaps, adjusting the data itself may help; such as removing some of the features or not segregating the rare occurrences into “Other” category.

## Context

The current analysis is that of a funding dataset. The task includes analysing the data to predict if an applicant will be successful in its proposed goals if they are rewarded funds by the donor organisation, Alphabet Soup.

The approach to predicting this includes analysing various aspects (called features) of previous applicants and deriving any relationship between these features and the status of their application, i.e. whether they were successful or not. If a relationship exists between the features and the status of the application, we can predict the outcome of future donations based on the features data of the application.

In this particular analysis, the dataset consists of 34,299 applications and their outcomes, along with following aspects (features) for each application: EIN (or a serial number), Name of the applicant, type of application, affiliation, classification of the application, what the funds were used for (use case), organisation, status, income amount, special considerations and ask amount. However, for analysis, the serial number and name have been dropped since they play no part in determining the results.

**Thus there are 9 features in this dataset. While the label is the Is\_Successful column.**

As part of the analysis, the categorical data of the features were broken down into Boolean values and we ended up with 43 feature columns.

### Step1: Preparing the Data

Before we can input the data into the model, we must prepare it for analysis. This includes, importing the .csv into python, reading it and then using pandas to convert it to a pandas dataframe. Two unnecessary columns, the EIN and Name were dropped as they were not important in our analysis.

Next we need to ensure the data is numerical. All categorical data is first encoded into numerical or Boolean values. In this analysis, many columns were categorical, as gauged by the dtypes and info functions. I converted this categorical data into Boolean values using the get\_dummies and encode functions.

### Step 2: Splitting the Data

Next, we need to split the dataframe into features and labels. Features are the various aspects of applications as mentioned earlier, while the labels are the outcomes, the Is\_Successful column.

### Step 3: Testing and Training sets

Once the data is split into features and labels, it has to be further split into testing and training sets. For all supervised ML methods, the algorithm has to be trained with data whose outcome is already known to us. Typically, 75% of the data is used for train the model, while 25% is used to test the model, to check if the model indeed works.

To split the data into testing and training sets, scikit-learn module of python has a train\_test\_split function which makes it convenient to split the data. The interesting feature about this function is that we can specify the way the data is split by using the stratify parameter. This parameter was used to make sure the data is evenly distributed across the testing and training datasets. So, out of ~34,300 rows, 18,200 had successful outcomes whereas 16,000 were not successful. The testing and training datasets are distributed so that this distribution is maintained in them. A scalar instance is then created to scale the data.

### Step 4 & 5: Compile, train and Evaluate Model

Once the data is split into testing and training sets, the model is initialised. A keras sequential model has been used since we want our data to be analysed in a sequential manner.

The model initially used contained the following:

* Input layer consisting of 9 neurons, and 43 input dimensions, one for each feature column.
* First hidden layer consisting of 9 neurons, the input dimensions don’t need to be specified here as they are the same as the number of neurons in the previous layer, i.e. 9
* Output later consisting of one unit since we expect a binary outcome.

It is them compiled, a callback function in initialised to save model weights every 5 epochs and then the training dataset is fed into it.

Initially, 50 epochs were chosen, which means the model trained and re-trained 50 times, each time adjusting itself as it “learned” from the previous epochs.

**At the end of the 50th epoch, the loss measured was 54.53% while accuracy was 73.53%**

Loss is a measure of how well a machine learning model is performing on a dataset. It quantifies the difference between the predicted values of the model and the actual ground truth values in the dataset. So the aim of our training should be to reduce the loss as much as possible.

I used theBinary Cross-Entropy loss function as it is used for binary classification tasks.

On the other hand, accuracy score is the overall correctness of the model’s predictions. The goal is to maximise the accuracy score.

The model was then tested using the test data.

**With test data, the loss measured was 55.62% while accuracy was 72.13%**

This comes close, but is not good enough of a score for the model to be used in predicting the if an applicant will be successful if given the funds by Alphabet Soup.

So, a model optimisation method was used to further optimise the model.

### Step 6: Optimise Model

The model was optimised using Keras-tuner. There are various permutations and combinations that can be used in creating a Neural Network model, including the number of neurons in each layer, the number of hidden layers, activation functions and epochs. Even the data that is fed into the model plays a crucial role in model performance. Keras-tuner is a keras framework that helps optimise the model by trying out a set of combinations of various parameters and figuring out which fits best for our data.

A separate file was created for the optimisation task. Other than the optimisation code itself, a feature of this file is a very basic EDA, or Exploratory Data Analysis performed on the data.

**From the EDA, I determined that the number of successful applications are: 18,261 (~53% of the total data) and the number of unsuccessful applications are: 16,038 (~47% of the total data).**

**I used three activation functions in the optimisation model: relu, tanh and sigmoid.**

**The number of neurons in first layer ranged between 5 and 20 with three steps; which means the first iteration was with 5 neurons, second was with 8 neurons, third was with 11 and so on. This was done to reduce computational burden on the model.**

**The number of hidden layers ranged from 1 to 6 and number of neurons in them ranged from 5 to 20, with three steps.**

**The optimised model provided an accuracy of ~72% with activation function tahn, 5 neurons, 3 hidden layers and 7 epochs.**

**Thus, even with an optimiser our data was model was not giving better results.**

Perhaps, adjusting the data itself may help; such as removing some of the features or not segregating the rare occurrences into “Other” category.