Deep Learning Challenge – Charity Funding Predictor

- ## Overview

The goal of this project is to configure a tool that can help the nonprofit foundation select the applicants for funding with the best chance of success in their ventures.

Using my knowledge of machine learning and neural networks I have created a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

The data source represents a CSV file containing more than 34,000 organizations that have received funding from Alphabet Soup over the years.

- ## Data preprocessing

The dataset includes several columns that capture metadata about each organization.

Two columns were removed from the data set – “EIN” and “NAME”, these cannot be considered target or feature.

The column “IS_SUCCESSFUL” is chosen as a target, it shows if money were effectively used.

The rest of the columns have been chosen as features.

Because some of the categorical features included uncommon categories that are rarely statistically significant the bucketing process has been applied. All rare values of “APPLICATION_TYPE” and “CLASSIFICATION” columns were combined into a single “Other” category.

For subsequent modification, the following techniques have been applied:

- Encoding
- Split into training and testing datasets.
- Scale and fitting

- **Compile, Train, and Evaluate the Model**

Initial Attempt (Models/AlphabetSoupCharity.h5):

The first attempt resulted in an accuracy score of 73.82%. This means that 73.82% of the model's predicted values align with the dataset's true values. That is a bit higher than the accuracy of the tested data (72.82%).
The activation function for the hidden layers is ReLU, as the modern default. And for the output layer – Sigmoid, as less complex, and suitable for binary classification type of prediction problem.

The initial number of neurons in the input layer is approximately equal to the number of inputs. And the structure was chosen to be pyramidal.

The hyperparameters used were:

- Input_dim = 43
- Layers:
  - layer1 = 40 neurons : activation function = 'relu'
  - layer2 = 20 neurons : activation function = 'relu'
  - layer3 = 1 neuron : activation function = 'sigmoid'
- epochs = 50

```
Epoch 49/50
804/804 [==============================] - 2s 2ms/step - loss: 0.5395 - accuracy: 0.7377
Epoch 50/50
804/804 [==============================] - 1s 2ms/step - loss: 0.5391 - accuracy: 0.7382
```

First Optimization Attempt (Models/AlphabetSoupCharity_Optimization1.h5):

Observing the progress of the initial attempt I considered increasing the number of epochs to 100, giving more chances for neuron weights to update. During the calculus the model was generating higher accuracy in the later epochs. In the review I also considered that the higher increase would lead to overfitting and will learn the noise instead of general pattern. Overall, this optimization increased the accuracy very insignificantly.

The hyperparameters used were:

- Input_dim = 43
- Layers:
  - layer1 = 40 neurons : activation function = ‘relu’
  - layer2 = 20 neurons : activation function = ‘relu'
  - layer3 = 1 neuron : activation function = ‘sigmoid’
- epochs **= 100**

```
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5374 - accuracy: 0.7384
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5371 - accuracy: 0.7388
```

Second Optimization Attempt (Models/AlphabetSoupCharity_Optimization2.h5):

For this optimization attempt I considered adding one hidden layer and increasing the number of neurons. In the output the accuracy increased to 74%.

The hyperparameters used were:

- Input_dim = 43
- Layers:
  - layer1 **= 80** neurons : activation function = ‘relu’
  - layer2 **= 40** neurons : activation function = ‘relu'
  - **layer3 = 20 neurons : activation function = ‘relu'**
  - layer4 = 1 neuron : activation function = ‘sigmoid’
- epochs = 100

```
Epoch 99/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5334 - accuracy: 0.7403
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5331 - accuracy: 0.7401
```

Third Optimization Attempt (Models/AlphabetSoupCharity_Optimization3.h5):

In the final attempt I decided to adjust the input data to ensure the class separation is not too narrow and valuable data is not neglected. I extended the number of bins for the "CLASSIFICATION" column from 6 to 16. That allowed the model to increase the training accuracy to 74.28%, testing accuracy remaining at 72.85%

The hyperparameters used were:

- Input_dim **= 53**
- Layers:
  - layer1 = 80 neurons : activation function = 'relu'
  - layer2 = 40 neurons : activation function = 'relu'
  - layer2 = 20 neurons : activation function = 'relu'
  - layer3 = 1 neuron : activation function = 'sigmoid'
- epochs = 100

```
Epoch 99/100
804/804 [==============================] - 1s 2ms/step - loss: 0.5282 - accuracy: 0.7427
Epoch 100/100
804/804 [==============================] - 2s 2ms/step - loss: 0.5279 - accuracy: 0.7428
```

- **Summary**

The final optimization gave the best result of 74.28% accuracy, this has been achieved due to input data update. This underlines the importance of data preprocessing.

Using this model, a successful applicant can be predicted with 73%-74% accuracy. The result can be improved most likely with an increase of the training data rather than optimization technics.