**Deep Learning Challenge - Creating a deep learning model to predict success of when given funding.**

**Overview of the analysis**

The purpose of this analysis was to develop a deep learning model to predict the success of charitable organizations funded by Alphabet Soup. The goal was to create a binary classifier that could determine whether applicants will be successful if funded, based on various features provided in the dataset.

**Data Pre-processing**

Q1. What variable(s) are the target(s) for your model?

The key target variable from the dataset provided is the IS\_SUCCESSFUL variable i.e. is the organisation successful when given funding in its desired outcomes. This is a binary indicator of success, with success given a value of 1, and those that are unsuccessful given a value of 0.

Q2. What variable(s) are the features for your model?

The feature variables given in the dataset to use to make predictions were:

* Application type using a T code,
* Affiliation (if any) to Alphabet Soup
* Classification using a C code
* Use Case e.g. for healthcare, preservation
* Organisation type e.g. trust, co – operative
* Status i.e. active (1) or inactive (0)
* Income amount in bands from 0 up to $50 million and above
* Whether there were any special considerations - a Yes / No
* The amount requested

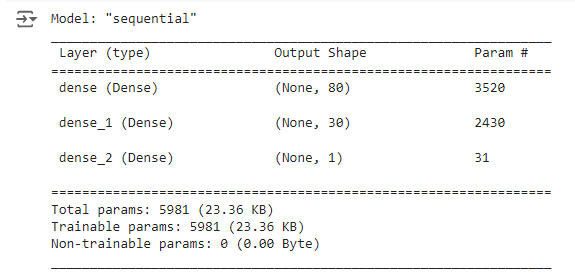
Q3. What variable(s) should be removed from the input data because they are neither targets nor features?

Initially the EIN and the name of the organisation requesting funding was removed, as these had no impact on determining whether the award of funding would lead to success of a venture. Whilst they are a useful identifier of specific cases, they are not required for predicting success.

**Compiling, Training, and Evaluating the Model**

Q4. How many neurons, layers, and activation functions did you select for your neural network model, and why?

Initially I used two hidden layers; with 80 neurons for the first, 30 neurons for the second. ReLU was the activation function selected. These gave the model answer provided i.e. the output shown below matched the guide that was given, with e.g. the 80/30 split being given.



The purpose of having it set up this way is that the two layers create a deep neural network, and allows the model to learn hierarchical features, with the first 80 neuron layer learning lower level features, and a second 30 neuron layer to learn more abstract, higher level features. Starting off with 80 neurons in the first hidden layer gives an initial wide representation of the data, and by reducing to 30 in the second makes the network to compress the information and to prioritise the more pertinent features. ReLU as an activation function allows the network to learn more complex patterns.

The effect of this structure is to allow the network to:

* Capture various aspects of the charitable organizations (first layer)
* Combine these aspects into more meaningful indicators of success (second layer)
* Make a final prediction based on these learned indicators (output layer)

Q5. Were you able to achieve the target model performance?

The accuracy of the model was 0.7262 - or 72.62%. This is clearly below the 75% accuracy required.

Q6. What steps did you take to increase model performance?

To increase performance, an iterative approach i.e. trial and error was taken. To speed up processing time - the original code took over an hour to run - I changed a couple of things to reduce this time - namely I increased the batch size from 1 to 100 so it rather than for each Epoch running through over 25,000 records, it runs through 250 records. This cuts the time down from over an hour to a couple of minutes. My plan is to use this as a base line, change e.g the number of neurons to see whether the accuracy increases – and if I find any changes that improve the accuracy of the model then apply them with the original batch size.

Original model with a Batch Size of 100 had an accuracy of 72.97%

I then doubled the number of neurons in each layer i.e. 160 and 60 neurons. This gave an accuracy of 73.00 – which is an increase on our base.

I then asked whether doubling was enough, so I quadrupled them – 320 and 120 neurons per layer to see what that did. However the accuracy dropped slightly to 72.9. The initial increase in accuracy is due to the increase in capacity for it to learn more complex patterns, capture more nuanced data and give a slight increase. However, further increases past this point suggest that by increasing the neurons to such an extreme gives either overfitting or an increase in complexity making the optimisation process more difficult.

Therefore, for an optimised solution, we will stick with the doubled as opposed to the quadrupled amount of neurons.

I then added to the original model an extra layer – so there were 3 layers with 160, 80 and 30 neurons in each. This gave an accuracy of 72.84, which is a slight decrease. I also looked at it from the other end – with 3 layers of 80, 30 and 15 neurons in each, which also gave a slight decrease in accuracy – down to 72.87.

Therefore it does seem that adding an extra layer – regardless of at which end of the layers you add – it does not seem to improve accuracy.

I then changed the activation function for the two hidden layers from reLU to seLU. This gave an accuracy of 72.58. I then looked whether I adjusted just one of the hidden layers rather than both – and neither increased accuracy. When the first hidden layer was changed to seLU, the accuracy dropped to 72.35, and when the second hidden layer was changed to seLU, it dropped to 72.86.

Therefore, for an optimised solution, we will stick with both hidden layers having an activation of reLU.

I then decided to alter the learning rate. Adam – the optimiser used – has a base learning rate of 0.001 so I decided to adjust this. I found that by changing the learning rate to 0.005 increased the learning rate the most – up to 73.18. The accuracy increased when the learning rate was also increased from 0.001 to 0.002 etc – and it peaked at 0.005, and higher learning rates i.e. 0.006, 0.007 showed a drop off of accuracy.

I then decided to increase the number of epochs from 100 to 200, to give more training time with a potential for better performance. However, this was not reflected in accuracy – it dropped slightly to 72.89. I then tried to decrease the number of epochs from 100 to 50 – which again dropped the accuracy down to 72.48.

So I have tried several methods to increase accuracy – increasing the amount of neurons, increasing the amount of hidden layers, changing the activation function, changing the activation rate and changing the number of epochs.

Only two were successful in increasing the accuracy. These are doubling the amount of neurons in both hidden layers and increasing the learning rate to 0.005. I then decided to combine these two features together to see whether by applying both, we get a combined improvement in accuracy. However the accuracy dropped to 72.24 – which was lower than doing each of these processes individually.

Therefore in terms of optimisation, the most effective was increasing the learning rate to 0.005 – increasing accuracy from 72.97 to 73.18.

This was done on a batch size of 100 – as I was looking for general trends to see what caused the improvements in a faster processing time. I then ran the code again with a batch size of 1, and this gave an accuracy of 72.23 – an improvement – but not a great one.

**Summary and recommendations.**

Architecture sensitivity:

* Base model: 2 layers (80 and 30 neurons) with ReLU activation
* Doubling neurons: Minimal improvement (73.00% vs 72.97%)
* Adding an extra layer: Slight decrease in accuracy (72.84%)
* Changing to SELU activation: Decrease in accuracy (72.58%)

Learning rate impact:

* 0.005 learning rate: Best performance (73.18%)
* 0.001 learning rate: Slightly lower performance (72.97%)

Overall performance:

* The best accuracy achieved was 73.18%
* Changes in architecture and parameters resulted in relatively small variations in accuracy (range: 72.58% - 73.18%)

Given these results, I would consider using a Random Forest model, which could provide:

* Stability across configurations: The neural network showed sensitivity to architectural changes and parameters. Random Forests are often more stable across different configurations, potentially requiring less fine-tuning.
* Automatic feature interaction: The neural network's performance did not improve significantly with additional complexity. Random Forests automatically capture feature interactions, which might be beneficial if our data has complex relationships not easily learned by the neural network.
* Handling of non-linear relationships: While ReLU and SELU can model non-linear relationships; the Random Forest's tree-based structure might capture different types of non-linearities present in the data.
* Potential for initial better performance: Given that the neural network's best performance was around 73%, there is the possibility that a Random Forest could achieve better accuracy from the start without extensive tuning.
* Feature importance: Random Forests provide feature importance measures, which could offer insights that the neural network does not readily provide.