# **Computational Intelligence**

## Gelmis Bartulis (B00080902)

## Assignment 2 - 20% 2018

The main goal of this assignment is to construct a multilayer neural network using TensorFlow software developed by Google. The target, was to analyse the program so it would be producing a result with a minimum of 0.01% error rate.

The program adjusts its weights accordingly to the provided data and defined variables. The data was trained by having a set of 500 samples and dedicating 100 samples to the evaluation and test data. The data was provided through Moodle page containing two different files, one of them being *train.cvs* and the other *eval.cvs.* These files were written in Excel to fit the data.

### Main processing

When the program was firstly launched, several error occurrences were observed and fixed before proceeding further. When all the errors were fixed, testing began using the training data.

### Initial observation

The program contained several different parameters which were defining the data processing and evaluation. Then input and output files were defined which took the training data in and when it was processed though, the end was put into the output folder.

Later, several arrays are seen which were filled with the variables from the datasets. Then different layers of placeholders were defined. After, two variables were initialised which were weights that would theoretically should adjust themselves accordingly to the inputs and variable entries. Then two Bias nodes were created, which were monitoring how flexible the perception is. Thus, providing a more accurately predicted data.

Then a hidden layer was defined with Sigmoid method, which took the weights and the bias that have been defined as parameters for adjusting the weights.

Finally, initial cost was created to see how expensive the algorithm is and how efficient it can get depending on the variables adjusted.

After all the base values initialised and enabled, session is created, which allowed the parameters to be persisted and adjusted within that session. The training data was trained the number of epochs that have been defined at the very start of the program as variables. The data was then printed which showed the hypothesised value, the target and the error occurrence between the values. Finally, when the for loop has finished, the answer has been created which was looking at the hypothesis for the Y - axis. By that accuracy was then formed by casting the answer as a float value.

When the training finished the session would then be saved into file for usage within the evaluation processing.

The initial processing had a simple and easy readable architecture, containing two input nodes, two hidden nodes and one output node (Fig. 1). Producing very low and inconsistent accuracy with a high cost of running the algorithm (Fig. 2).

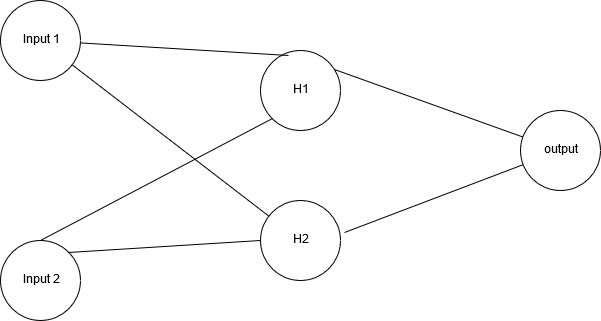


Figure : Shows the initial NN architecture

At the very beginning of processing, the application was given the following results (Fig. 2). Which showed that the cost was quite high, and the weights were not adjusting correctly. Therefore, some specific adjustments had to be made.

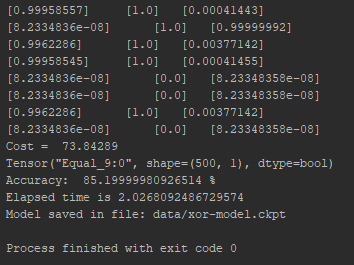


Figure : Initial results

The initial program was focusing only on four variables, making it easy and in - efficient to use since such algorithms are mostly used for large heaps of data (Fig. 3).

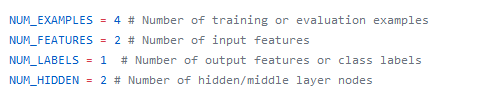


Figure : Shows initial program setup

### End result

The variables have been adjusted to structure the dataset and make it as accurate as possible. As seen in Fig. 4, the following variables have been set to produce he most accurate outcome.

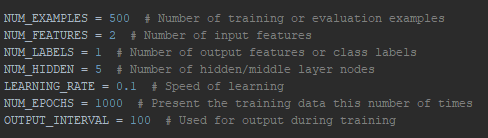


Figure 4: Final variables

In order to retrieve the mostly accurate answer, the program was run 10 times and the average value has been summed up, by looking at each of the runs and creating a mean of the accuracy. Which shown to have 100% accuracy – according to the process. Although, the cost changes every time the program is executed.

The program architecture can be observed in the figure below. As from Figure 4, the diagram has been setup to show the structure and the process of the program that has been adjusted to lift the variables automatically and to show the difference in processing. Where the program still contains two inputs, it has been observed that the program works best with five hidden nodes that help to increase the overall accuracy (Fig. 5).

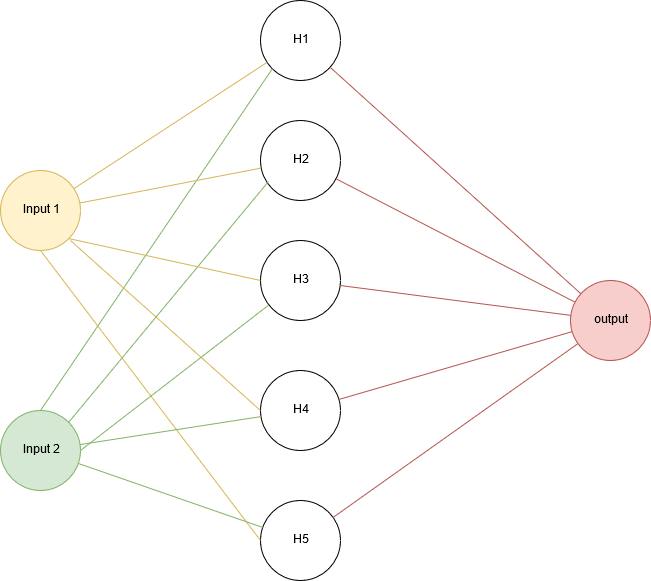


Figure 5: Shows the architecture of the adjusted program

Several tests have been conducted in order to receive an approximate answer that would be more accurate than from judging a single result. Therefore, the cost, average error, run time and accuracy have been recorded in order to get the most accurate values as seen from the table below (Fig. 6).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Cost** | **Accuracy** | **Time (s)** | **Average error (%)** |
|
| Run 1 | 1.3652592 | 99.80% | 2.009366986 | 0.00194665 |
| Run 2 | 0.08447702 | 100.00% | 2.159825161 | 0.000689955 |
| Run 3 | 0.04794546 | 100.00% | 2.005042696 | 0.000343294 |
| Run 4 | 0.07118841 | 99.80% | 2.333617657 | 0.003509502 |
| Run 5 | 0.063744925 | 100.00% | 1.961200634 | 0.000229638 |
| Run 6 | 0.05665965 | 100.00% | 2.058145624 | 0.004143752 |
| Run 7 | 2.2385926 | 99.60% | 2.102034364 | 0.000496871 |
| Run 8 | 0.09012162 | 100.00% | 2.123225137 | 0.001160356 |
| Run 9 | 0.050310403 | 100.00% | 1.963116625 | 0.000508125 |
| Run 10 | 0.054663517 | 100.00% | 1.967756358 | 0.00306439 |
| **Averages** | **0.412296281** | **99.92%** | **2.068333124** | **0.001609253** |

Figure 6: Running accuracy