**BIOLOGICALLY INSPIRED COMPUTATION**

**F20BC**

**ASSIGNMENT 1**

**Yola Jones**

**H00152432**

**Link to Github:**

**https://github.com/yj48/F20BC-Bio-Inspired/**

**Included on Github:**

* Matlab script for MY\_OPTIMIZER.m and relevant functions, as well as script provided by COCO
* This report
* A README.txt file explaining how to run code and generate different graphs
* A word document containing graphs of iterations against fitness for each test completed
* An excel document containing average values and analysis of data

This report will detail the process of building a Multi-Layer Perceptron Neural Network, with weights evolved by a Genetic Algorithm. This network is put into the COCO benchmarking software to test how well it can evolve to replicate a set of five functions. Three variables have been refined for greater accuracy in results, and the neural network as a whole will be analysed against the five functions used by comparing their average fitnesses.

There were a number of variables to be considered when creating the Neural Network. The predominant variables were the number of hidden layers and the number of nodes in each layer. The structure of the network was specified in the coursework description, the MLP requiring all nodes in one layer to be connected to all nodes in the following layer, with no feedback. Other factors to consider were the activation function and the bias at each neuron. The logsig activation function was quickly decided on with no bias, the lack of bias reducing complexity and the logsig function proving to be successful in basic initial tests.

After doing some research into which structures were best for the neural network, it was found that a simple structure was usually sufficient, with most sources[[1]](#footnote-1) suggesting one hidden layer was enough. The number of nodes in that layer then lies between the number of input nodes (DIM) and number of output nodes (one).

In the initial stages of the project, getting a neural network working was made a priority over using theory to back up the structure. Because of this the structure was initially set to DIM-5-2-7-5-1, with DIM being the number of input nodes (a number which is passed into the function from COCO), four hidden layers with 19 total hidden nodes, and one output node.

Once this was working it was decided to test the network with a DIM-5-1 structure. This was far more in line with the theory and therefore was expected to generate the best results. Since the theory being looked at was not largely backed up with experimental results, a DIM-2-2-1 topology was tested as well, to prove or disprove conjecture.

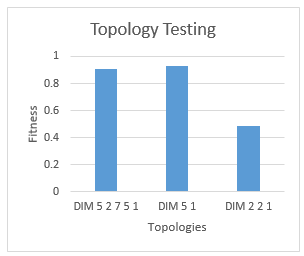
Having got the algorithm working in COCO (including the genetic algorithm which will be discussed later), the topologies were tested with the first 5 functions. The results are a representation of the average fitness for each topology, across all 5 functions. More detail and the exact values can be seen in the excel document uploaded to github, showing all generated results and graphs.

Figure 1: Testing Topology with first 5 Functions

As can be seen from Figure 1, the topology that produced the best results overall was the DIM-5-1 structure, closely followed by DIM-5-2-7-5-1. These results support the theory described above. Complex topologies like the first one are expected to generate good results, but the computational complexity isn’t needed in most applications. DIM-5-1 is simple enough to produce computationally efficient yet accurate results. The DIM-2-2-1 topology performed the worst as expected, the results proving the theory to be accurate for this application.

The weights used for the neural network are determined by the genetic algorithm. The GA generates a 1D array of weights, which is then concatenated by a separate function into matrices, where each matrix corresponds to a different layer. So for example, the solution [w13,w14,w23,w24,w35,w45] generated by the GA would become the solution a = and b = according to Figure 2.

w­13 w14

w23 w24

[ ]

w­35

w45

[ ]

Figure 2: Neural Network Weights Example

a

b

The main variables for the genetic algorithm are the crossover method and rate, the mutation method and rate, and the number of solutions in each population. The crossover and mutation methods were decided early on, using the lecture notes as a guide. A two-point crossover method was proposed, since it had been used for previous courseworks and had proven successful. The M-random-gene new-allele mutation[[2]](#footnote-2) was used, and the new population is made up as follows:

* Half is made of solutions generated by two-point two-parent crossover
* Two of the solutions are made up from mutations
* Two of the solutions are made up of the best two solutions from the previous generation
* One of the solutions is the worst solution from the previous generation
* The rest of the population is made up of random members of the previous generation

It was decided to incorporate elitism as sources[[3]](#footnote-3) suggested this would generate better results. The worst solution of the previous generation was added to test what it would do to the fitness.

The fitness came from a combination of the Neural Network and COCO. To test the fitness a random set of DIM inputs are generated in the range -5 to 5. Those inputs are put through both the Neural Network and fed into COCO through feval(FUN,xpop), with COCO generating the expected output and the Neural Network generating the actual output. These are then compared, and 1/(the difference between the two) become the fitness of that set of weights, with greater fitnesses meaning better results.

The next thing to test was the number of solutions in one population. This was initially set to 20, a completely arbitrary number. Since this was a very simple parameter to test, it was decided to rely on experimentation rather than theory to determine which value would be used. The values were tested in the same way the topology was tested, taking three iterations of each of the first five functions, and generating a complete average for each value to be tested.

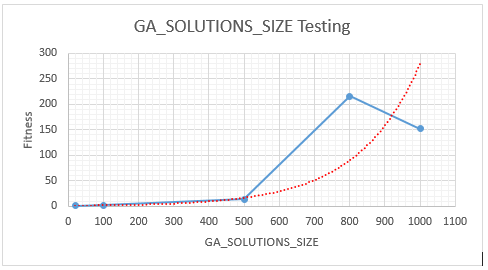


Figure 3: Testing GA\_SOLUTIONS\_SIZE for the First 5 Functions

Initial conjecture led to the belief that the data would follow the red path, as shown in Figure 3, with larger values giving better results but following an exponential curve. The data generated shows this to be initially the case, but the results generated at 800 and 1000 completely disproving this theory. This is suspected to be due to luck, i.e. the population at 800 being initially extremely fit compared to the population at 1000. Taking an average of a greater number of tests should theoretically negate this jump, but more tests would need to be done. Despite the jump at 800, the program was found to be extremely slow in generating data due to the high number of processes required. A value of 500 was picked to carry out further tests, as it produced a balance of accuracy and efficiency.

To implement the GA to the Neural Network multiple functions were used to increase readability and to make debugging far easier. The neural network code was written to be extremely user friendly, declaring four user alterable variables at the beginning of the code in capital letters. The structure of the neural network declaration was also made extremely simple, with NODES being a 1D matrix of seven values. Each index represents a layer and the number contained in that index being the number of nodes in that layer, e.g. the declaration NODES = [1,2,5,6,0,0,0] represents a topology with one input node, two hidden layers with 2 and 5 nodes respectively, and six output nodes. The number of weights required, the output layer, the size of each chromosome in the GA and the neural network iteration all come off this one matrix, allowing quick changes to the topology to be made.

The final value to be tested was the TRAINING\_ITERATIONS. This denotes the number of times the neural network is trained for each function. The value was initially set to 200 and the results were generated in exactly the same way as before:

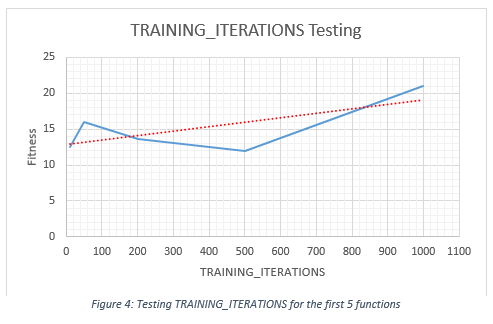


Figure 4: Testing TRAINING\_ITERATIONS for the first 5 functions

These results were the biggest surprise of all. The result was expected to be fairly linear as shown in the red line in Figure 4, but the result was much more jumpy than predicted. The overall result does tend upwards, but isn’t consistent in first 500 values. 50 was therefore decided on to take forward, and was used in the final analysis.

For the final test it was decided to make the topology a DIM-5-1 network, with 800 for GA\_SOLUTION\_SIZE and 50 for TRAINING\_ITERATIONS. Despite taking a while to compute it took far less time than previous tests, whilst still generating a successful run. The overall result of this test for each of the five functions can be seen in Figure 5.

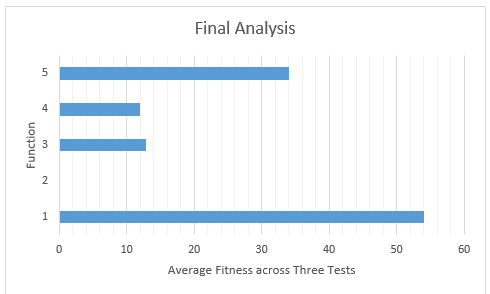


Figure 5: Final fitnessess generated with optimal values across 5 functions

Having completed a final run, all results obtained were used to find which of the functions the neural network was most successful at replicating. This was done by taking the average fitness of each result generated, for each of the five functions. The results are as follows:

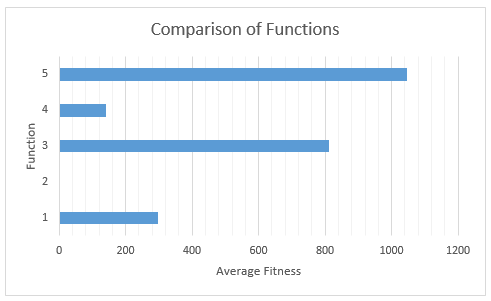


Figure 6: Comparison of Functions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| FUNCTIONS | 1 | 2 | 3 | 4 | 5 |
| AVERAGE FITNESS | 296.20 | 1.84 | 811.27 | 141.78 | 1045.92 |

Table 1: Numerical Average Fitness Values

As can be seen from the Figure 5 and Table 1, function 2 (the *Ellipsoid separable with monotone x-transformation function*) performed the worst with the algorithm, while function 5 (*Linear slope, neutral extension outside the domain*) performed overwhelmingly the best. This is suspected to be down to the complexity of each of the functions, and that different topologies with more hidden layers may give a completely different result. The results from analysing different topologies (the numerical results of which can be seen in the spreadsheet alongside this report) appear to support this, but more tests would need to be carried out to confirm this definitively.

Overall the results were generally as expected, with a few surprises. The results from the function comparison are consistent throughout, and an optimal solution for the three variables to be tested has been reached.

1. *Multi-layer perceptron (MLP) architecture: Criteria for choosing number of hidden layers and size of the hidden layer?* (2016) Available at: http://stackoverflow.com/questions/10565868/multi-layer-perceptron-mlp-architecture-criteria-for-choosing-number-of-hidde (Accessed: 10 October 2016). [↑](#footnote-ref-1)
2. Vavak, F. and C. Fogarty, T. (1996) ‘Comparison of steady state and generarional GA’, *IEEE*, [↑](#footnote-ref-2)
3. Haykin, S. (2008) *Neural networks and learning machines: A comprehensive foundation*. 3rd edn. Harlow: Prentice Hall. [↑](#footnote-ref-3)