F20BC Biologically Inspired Computation
CW1 - Black Box Optimisation Benchmark (BBOB)

Kyna Mowat-Gosnell, km362

Introduction to MLP-GA hybrid approach

Programming Language of choice for this assignment was MATLAB because it has extensive matrix and graphing capabilities within the software that the assignment brief made clear would be implemented.

Implementation

Implementing the Neural Network proved to be extremely difficult and there were several different implementations I pursued before the final version. My initial attempt was to split up the parts of the implementation and program them separately by the following sections:

Implement output and class conversion functions:

1. outputToClass is a function that converts an output matrixNode to a classNumber vector (1d matrix)
2. classToOutput is a function that converts a classNumber vector back to an output matrix

Seeing as MATLAB is a relatively new programming language for me, here are details about the code:

In outputToClass, length: length of largest array **dimension.**

Example below:

```
V = [5 6 7 8 9 10]
L = length(V)
L = 6
```

In classToOutput, max: largest elements in array.

Example below:

```
A = [2.3 4.5 1.9, 1.3 1.57 2.45]
A =
        2.30    4.50    1.90
        1.30    1.57    2.45
[M,I] = max(A,[],2)
```

```
M =                    I =
        4.50                  2
        2.45                  3
```

Where M is for maximum and I is for index in the matrix

In both, zeros creates an array of zeros with the sizes that the variables within parentheses return. Example below:

sampleNumber = 3

max(classNumber) = 2

Z = zeros(sampleNumber, max(classNumber))

```
Z =
        0       0
        0       0
        0       0
```

If Z returns matrix that isn't all zeroes, it shows that there is a bug in the program

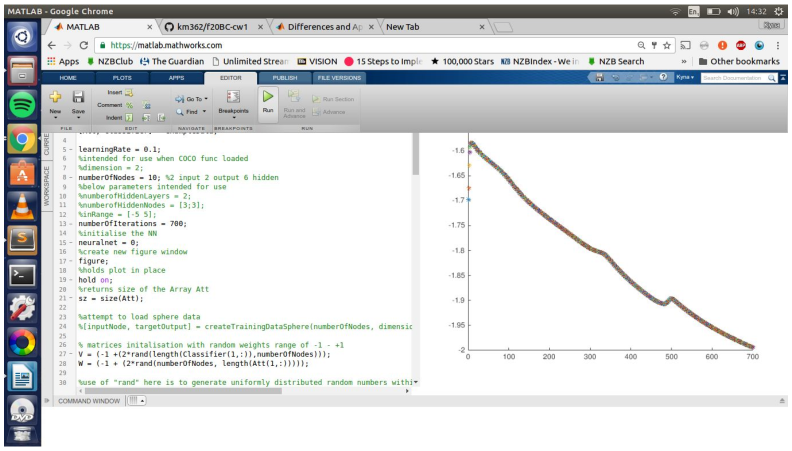
Final Implementation

At the start of the program, two weight matrices initialised with random weights in the range of -1 and 1. One of the matrices is the weight matrix between input and hidden layer, and one is matrix between hidden and output layer. I decided to use tournament selection for my EA because it is fairly simple to implement while being just as good as other more complex evolutionary algorithms. While initialising the population, using sort(A,2) allows the random population array elements to be sorted into ascending order in each row.

Fitness function uses variance (var(A,[],dim)) to find the variance of difference of population along the dimension, 2 in this case. Printing the statistics helps to make sure that we can keep a track of the program that it is doing what we want. This proved useful in debugging because at one point the EA was stuck in a loop, and so printing into the command window allowed the bug to be identified and corrected.

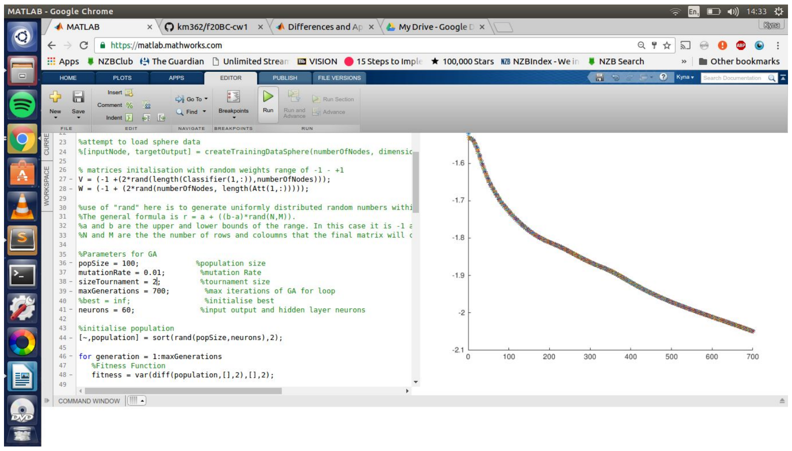
The example data that I have used to run my program isn't COCO, but something unrelated because I couldn't understand how to get COCO to work with my program.

Tournament selection works by running several tournaments among randomly chosen individuals within the population,which is determined by the sizeTournament parameter.



```
learningRate = 0.1;
%intended for use when COCO func loaded
%dimension = 2;
numberOfNodes = 10; %2 input 2 output 6 hidden
%below parameters intended for use
%numberofHiddenLayers = 2;
%numberofHiddenNodes = [3;3];
%inRange = [-5 5];
numberOfIterations = 700;
%initialise the NN
neuralnet = 0;
%create new figure window
figure;
%holds plot in place
hold on;
%returns size of the Array Att
sz = size(Att);

%attempt to load sphere data
%[inputNode, targetOutput] = createTrainingDataSphere(numberOfNodes, dimensic

% matrices initalisation with random weights range of -1 - +1
V = (-1 +(2*rand(length(Classifier(1,:)),numberOfNodes)));
W = (-1 + (2*rand(numberOfNodes, length(Att(1,:)))));

%use of "rand" here is to generate uniformly distributed random numbers withi
```

Fig 1: sizeTourament = 50

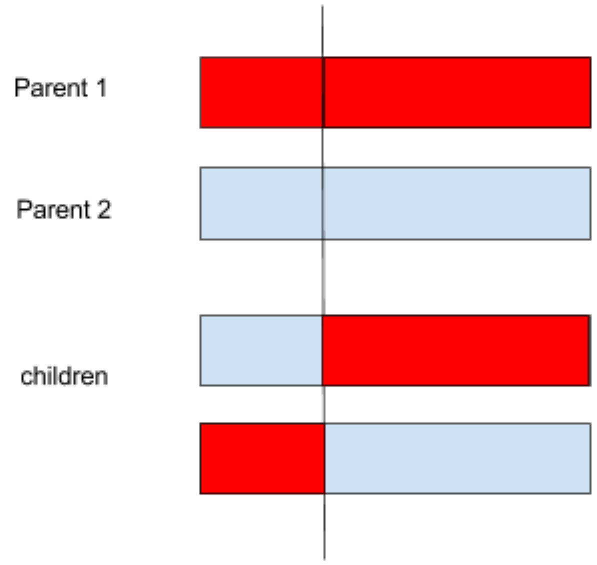


```
%attempt to load sphere data
%[inputNode, targetOutput] = createTrainingDataSphere(numberOfNodes, dimensic

% matrices initalisation with random weights range of -1 - +1
V = (-1 +(2*rand(length(Classifier(1,:)),numberOfNodes)));
W = (-1 + (2*rand(numberOfNodes, length(Att(1,:)))));

%use of "rand" here is to generate uniformly distributed random numbers withi
%The general formula is r = a + ((b-a)*rand(N,M)).
%a and b are the upper and lower bounds of the range. In this case it is -1 a
%N and M are the the number of rows and coloumns that the final matrix will c

%Parameters for GA
popSize = 100;              %population size
mutationRate = 0.01;         %mutation Rate
sizeTournament = 2;          %tournament size
maxGenerations = 700;         %max iterations of GA for loop
%best = inf;                  %initialise best
neurons = 60;                %input output and hidden layer neurons

%initialise population
[~,population] = sort(rand(popSize,neurons),2);

for generation = 1:maxGenerations
   %Fitness Function
   fitness = var(diff(population,[],2),[],2);
```

Fig 2: sizeTournament = 2

Fig 1 is 50 individuals in the tournament. Fig 2 is 2 individuals in the tournament. You can see that with a smaller tournament size the line is smoother, but this also shows that it is less representative of the population.

Single point crossover works by choosing a point along the string of 2 parents, and then swapping information after this point with the other parent. Here is a diagram:



Sub2ind is used in crossover and mutation, it returns index equivalents of specified subscripts for the dimensions of the matrices specified.

Neuralnet is held within a while loop and sorts out all the weights as they are propagated through the network. It keeps track of the weights. The weights are then adjusted before the loop increments so that the next time it runs it uses the new weights found.
The RMS error is used to evaluate the difference between the sample and the population so we can keep track of how the program evolves, and see that it is evolving with the RMS decreasing. The activation function I chose to use was the hyperbolic tangent activation function, as it is a translated version of the sigmoid function, and seemed slightly more accurate.

Bibliography

Uk.mathworks.com. (2016). MATLAB Documentation. [online] Available at: https://uk.mathworks.com/help/ [Accessed Oct.-Nov. 2016].

YouTube. (2016). Backpropagation Neural Network - How it Works e.g. Counting. [online] Available at: https://www.youtube.com/watch?v=WZDMNM36PsM [Accessed 23 Oct. 2016].

Hansen, N., Mersmann, O., Tusar, T., Auger, A. and Brockhoff, D. (2016). COCO: The Experimental Procedure. 1st ed. [ebook] Available at: https://arxiv.org/pdf/1603.08776v2.pdf [Accessed 1 Nov. 2016].

Panthimanshu17.wordpress.com. (2016). 06 | August | 2013 | Artificial Intelligence & Machine Learning. [online] Available at: https://panthimanshu17.wordpress.com/2013/08/06/ [Accessed 28 Oct. 2016].

Vallejo, M. (2016). This is lecture 6 of `Biologically Inspired Computing' Contents: Steady state and generational EAs, basic ingredients, example encodings / operators.