Biologically Inspired Computation

Coursework 1

Nicholas Robinson
Sam McNaughton

https://github.com/kyleNR/Genetic-Algorithm-Neural-Network-Evaluation

November 14, 2016

1 Experiment

1.1 Neural Network

1.1.1 Topology

The Neural Network was created to be able to change easily. This ultimately meant an input parameter being used to define the size of the input layer to allow it to calculate a function with any possible dimension size.

The final topology of the network was an input layer of size dimension, a hidden layer of size 4, another hidden layer of size 3 and an output layer of size 1.

1.1.2 Activation Function

The activation functions used in the network were switched around a lot until ultimately we decided to use the Bent Identity function to allow an output size from ∞ to $-\infty$ without scaling the output.

1.2 Genetic Algorithm

1.2.1 Data Representation

The GA (Genetic Algorithm) uses a list to hold the weights for the NN (Neural Network). Although this is different from the matrix structure used within the network, it allows the GA to be used on any topology of NN.

1.2.2 Selection

Tournament was the selection method chosen to be used in the GA. It takes 4 random chromosomes and finds the fittest out of the first two, and the fittest of the second two. With the two chromosomes to uses the crossover method to create a child to go into the population of the next generation. It was used as the selection method as it has been proven to be effective[1]. Although a larger amount of comparisons could be done to reduce the number of generations needed to reach a local minimum, it was decided to trade this to achieve a better time to run the program in.

Elitism is keeping the best chromosome of every generation and adding it to the next generation. It was added to the GA to allow it to reach a local minimum faster and also to stop the GA regressing if the population does not find a better fitness. While it is possible to use elitism to more than one chromosome between generations, it was chosen to only keep one, as it would keep the fitness from decreasing without cause the variety in chromosomes to stagnate.

1.2.3 Modification

The GA uses a Two-point crossover for its crossover function. This is a simple method, although slightly more complex as a One-point crossover method. It takes two points as indexes, takes the first parent up to that point and after the second point, then takes the second parent between the two points to create a full child. While this could have been used to create two children, it was decided to use this to only create one to create a larger degree of diversity in the population.

The mutation function is basic but effective at slight variations in the output. It takes the chromosome and loops randomly between 1 and up to the number of weights. In the loop it selects a random weight and multiplies it by either 0.75 or 1.25.

The mutation chance is the probability that the child created with the crossover function will be mutated before being added to the new population.

1.2.4 Generations

The generations is the number of iterations for the GA to run through. It is set as a default value of 200.

1.2.5 Population

The generations is the number of chromosome to use in each generation in the GA. It is set as a default value of 50.

1.2.6 Data Set

The size of the dataset used to train the GA is set with the DataGenerator file. Higher the number of sets of data the GA will train with, the more accurate it can be theoretically. In reality the accuracy is based much more on the Generation, Population, and Mutation Chance size. The larger the dataset, the longer the GA will take to train as each fitness test will take longer.

1.3 Neural Network Compare

The Experiment to compare the Neural Network against the COCO functions was set using the exampleexperiment.py file from COCO. This file will train the Neural Network based of the dimensions and functions given before running and comparing each pair of functions.

1.3.1 Functions

The functions chosen to compare against COCO were the Sphere, Ellipsoidal, and Rastrigin functions. These are represented in COCO with the function IDs of 1,2,3. Although only these were used, the Neural Network has the capacity to be compared on all 24 noiseless functions, if given the data.

1.3.2 Dimensions

The Dimensions chosen to compare the COCO functions with the NN are 2,3,5. The dimensions were chosen to keep the time to run the experiment down while also being compatible with

COCOs post-processing. The Neural Network can handle any size of dimension, but the time to evolve the weights will increase in relation.

2 Results

2.1 Finding Best Weights

2.1.1 Settings

Elitism provided the best values as it never regressed in fitness. When disabled, the population's fitness could and in most cases regress backwards in some generations.

2.2 Compare Functions

2.2.1 Parameters

Genetic Algorithm Parameters

- Generations = 200
- Population = 30
- Mutation Chance = 25
- Elitism = True
- Dataset Size = 50

Function Comparison Parameters

- Dimensions = 2,3,5
- Functions = 1-24
- Instances = 1-5, 41-50

2.2.2 Time

Running the NNCompare program took 1.34 hours to fully complete. This included the time spent Training the network for each dimension and function and running the Monte Carlo selection optimisation. More accurate results may have been attained by using higher parameters for generations, population size and dataset size but this would have resulted in the program running for longer.

2.2.3 Error

The error between functions varies quite a lot between the function being used to test. Originally the goal was to have an accuracy of 1 x 10^{-8}. This was quickly proven to be not possible for us.

The average error over the 3 dimensions and 24 function is 71673030.893774. This is calculated

Dimension Error Breakdown

- Dim: 2 - Average Error: 180162184.663644
- Dim: 3 - Average Error: 64892443.572706
- Dim: 5 - Average Error: 32345721.778468

Function Error Breakdown

- Function: 1 Mean Error: 164.804346
- Function: 2 Mean Error: 10896935.209143
- Function: 3 Mean Error: 1294.011928
- Function: 4 Mean Error: 1276.370219
- Function: 5 Mean Error: 375.322413
- Function: 6 Mean Error: 272930.882969
- Function: 7 Mean Error: 702.017186
- Function: 8 Mean Error: 114860.286929
- Function: 9 Mean Error: 56482.608131
- Function: 10 Mean Error: 15736200.667834
- Function: 11 Mean Error: 15228843.708646
- Function: 12 Mean Error: 733934812.030683
- Function: 13 Mean Error: 986.166639
- Function: 14 Mean Error: 190.375155
- Function: 15 Mean Error: 789.724851
- Function: 16 Mean Error: 206.320422
- Function: 17 Mean Error: 199.310447
- Function: 18 Mean Error: 267.512007
- Function: 19 Mean Error: 307.240868
- Function: 20 Mean Error: 47965.536369
- Function: 21 Mean Error: 251.182707
- Function: 22 Mean Error: 926.811691
- Function: 23 Mean Error: 121.188542
- Function: 24 Mean Error: 233.393122

References

[1] G. Rawlins, Foundations of genetic algorithms, 1st ed. San Mateo, Calif.: M. Kaufmann Publishers, 1991, pp. 78-82.