Program Structure and Algorithm

Final Project: Genetic Algorithm Repo: *https://github.com/shoutinglyu94/INFO6205_304.git*

Director: Robin Hillyard Author: Shouting Lyu 001822094, Chang Liu 001837020

Problem

Weight initialization can have a profound impact on both the convergence rate and final quality of a network and Genetic Algorithm is a very useful approach on NP hard problem. In this project, we created a genetic algorithm to improve weight initialization for back propagation neural network. The problem our neural network is trying to solve is a classical recognition problem on iris flowers. There are four decimal numbers as input data which represent the features of three types of iris flowers, and the trained model is used to recognize the type of an iris flowers. The data set is public in the UCI Machine Learning Repository: *https://archive.ics.uci.edu/ml/datasets/iris*.

Implementation Design

Genotype& Phenotype: The genotype and phenotype of our algorithm is basically equal and another way of implementing them in separate expression is still under experiment. The genotype includes two matrixes which is the weight of hidden layer and output layer, because the structure of our network is a 3-layer BP neural network. Given the number of input is X, the number of nodes in hidden layer is Y and the number of output is Z, then the first matrix is X×Y and the second matrix is Y×Z and each w_{ij} is a decimal number between -1 and +1.

$$\begin{pmatrix} w_{11} & \cdots & w_{1y} \\ \vdots & \ddots & \vdots \\ w_{x1} & \cdots & w_{xy} \end{pmatrix} \qquad \begin{pmatrix} w_{11} & \cdots & w_{1z} \\ \vdots & \ddots & \vdots \\ w_{y1} & \cdots & w_{yz} \end{pmatrix}$$

Fitness function: The fitness of one generation is closely relevant to performance of each model. We represented the performance of a model using variance between the output result and standard output. Hence, if we want to find the best solution, then it means we are going to find an individual which has the smallest variance. The fitness of generation is f(generation) = 1/ s(generation), and the s(generation) is the variance of a model. The best, worst, average fitness was calculated for each generation and was logged using Apache Commons Log and all jar files were also attached in the repository.

Mutation: We did a random mutation which randomly selected base of random length with a max-mutation-length of three. There is an example in which the mutation length is 2.

Original chromosome:

0.11	-0.2	0.27
0.32	0.4	0.45
0.18	-0.3	-0.3
-0.2	-0.22	0.11
-0.11	-0.1	-0.22
-0.16	0.13	-0.43
0.4	0.15	0.2
0.47	0.11	0.31

gene_in_weight(original)

0.4	0.5	-0.1	-0.2	-0.13	-0.1	0.2	0.35
0.2	-0.11	-0.05	0.08	0.11	-0.2	0.13	0.21
0.44	0.39	0.46	-0.1	-0.05	-0.38	0.4	0.39

gene_out_weight(original)

Gene to mutation:

0.11	-0.2	0.27
0.32	0.4	0.45
0.18	-0.3	-0.3
-0.2	-0.22	0.11
-0.11	-0.1	-0.22
-0.16	0.13	-0.43
0.4	0.15	0.2
0.47	0.11	0.31

0.4	0.5	-0.1	-0.2	-0.13	-0.1	0.2	0.35
0.2	-0.11	-0.05	0.08	0.11	-0.2	0.13	0.21
0.44	0.39	0.46	-0.1	-0.05	-0.38	0.4	0.39

gene_in_weight(to change) *gene_out_weight(to change)*

Chromosome after mutation:

0.11	0.14	0.27
0.32	0.4	0.45
0.18	-0.3	-0.3
-0.2	-0.22	0.11
-0.11	-0.11	-0.22
-0.16	0.13	-0.43
0.4	0.15	0.2
0.47	0.11	0.31

0.4	0.5	-0.1	-0.2	-0.13	-0.1	0.2	0.35
0.2	-0.37	-0.05	0.08	0.11	0.47	0.13	0.21
0.44	0.39	0.46	-0.1	-0.05	-0.38	0.4	0.39

gene_in_weithg(after change) *gene_out_weight(after change)*

Crossover: We randomly choose the starting point of crossover and the ending point of crossover and then perform a swap mutation based on the value of the other parent on these points.

Example:

Parent1:

0.11	0.14	0.27
0.32	0.4	0.45
0.18	-0.3	-0.3
-0.2	-0.22	0.11
-0.11	-0.11	-0.22
-0.16	0.13	-0.43
0.4	0.15	0.2
0.47	0.11	0.31

0.4	0.5	-0.1	-0.2	-0.13	-0.1	0.2	0.35
0.2	-0.37	-0.05	0.08	0.11	0.47	0.13	0.21
0.44	0.39	0.46	-0.1	-0.05	-0.38	0.4	0.39

Parent2:

0.38	0.44	-0.2
-0.11	-0.4	0.28
0.23	0.5	0.29
0.4	-0.42	-0.05
0.31	0.22	-0.43
-0.09	0.33	0.29
0.02	-0.1	0.47
-0.01	0.2	-0.09

0.13	0.52	0.46	-0.13	-0.13	0.28	-0.14	0.5
0.19	0.26	-0.3	0.01	-0.29	0.1	-0.12	0.33
-0.3	0.5	0.29	-0.11	0.37	-0.22	0.41	-0.5

Child1:

0.11	0.14	0.27
0.32	-0.4	0.28
0.18	0.5	0.29
-0.2	-0.42	-0.05
0.31	0.22	-0.22
-0.09	0.33	-0.43
0.4	0.15	0.2
0.47	0.11	0.31

0.4	0.5	-0.1	-0.2	-0.13	0.28	-0.14	0.5
0.19	0.26	-0.3	0.01	-0.29	0.1	-0.12	0.33
0.44	0.39	0.46	-0.1	-0.05	-0.38	0.4	0.39

Child2:

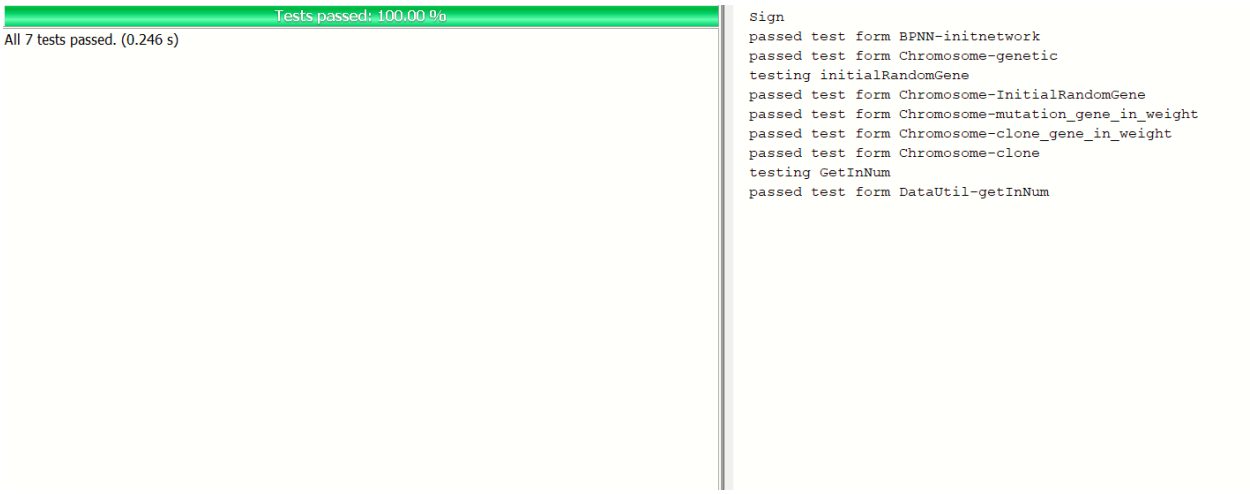
0.38	0.44	-0.2
-0.11	0.4	0.45
0.23	-0.3	-0.3
0.4	-0.22	0.11
-0.11	-0.11	-0.43
-0.16	0.13	0.29
0.02	-0.1	0.47
-0.01	0.2	-0.09

0.13	0.52	0.46	-0.13	-0.13	-0.1	0.2	0.35
0.2	-0.37	-0.05	0.08	0.11	0.47	0.13	0.21
-0.3	0.5	0.29	-0.11	0.37	-0.22	0.41	-0.5

Evolution: A single evolution includes processes of selection, crossover, and mutation. We used a Priority Queue to place the individuals and created a customized comparator class to maintain their order by the fitness. For each generation, we choose individuals and let them sexually reproduce using roulette method, and each child would experience mutation with a fixed probability 0.01, then we and kept the half of new individuals and another half is selected from the parent population.

Result

Our project builds and passes all the unit test cases:



We ran different experiments with 100 population for each generation and each model in an individual would be trained 100 times using the aforementioned dataset. The gene of the first population is randomly generated and is passed to the offspring by the genetic method. We did 4 trials to examine the performance of our algorithm. The first and the second trail were not set learning rate for neural network and the results are shown in Figure 1 and 2, the last two trials repeated the previous experiments but with learning rate 0.5 for weight and threshold and the results are shown in Figure 3 and 4.

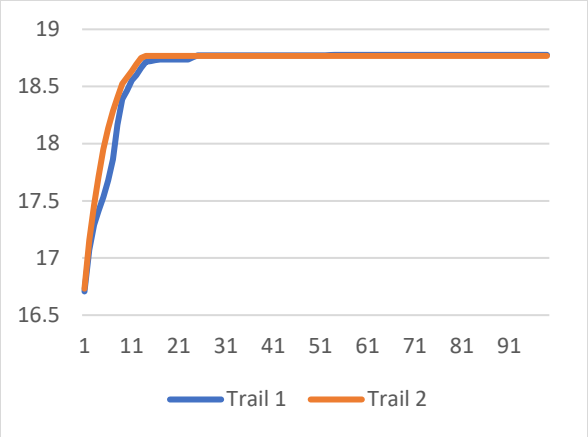


Figure 1 Average Fitness without learning rate

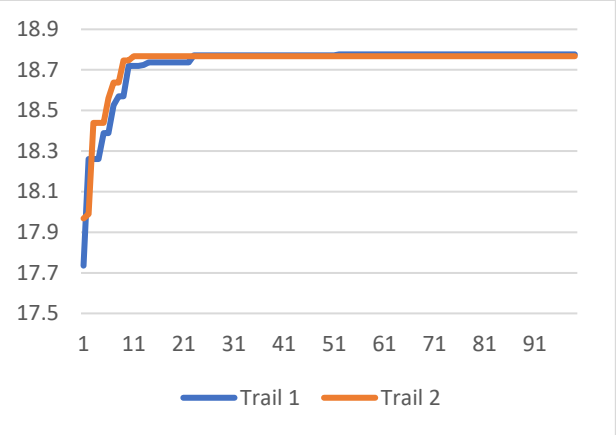


Figure 2 Best Fitness – Generation without learning rate

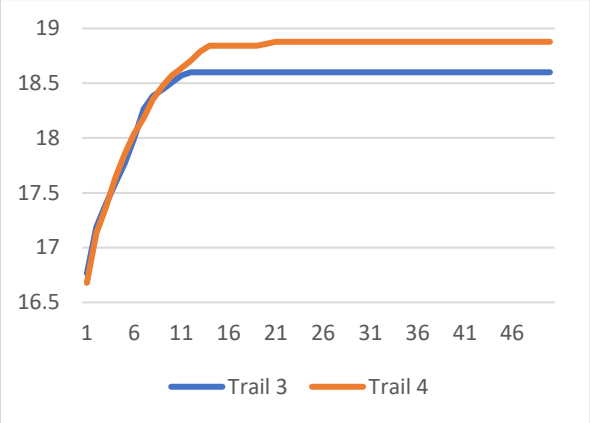


Figure 3 Average Fitness with learning rate

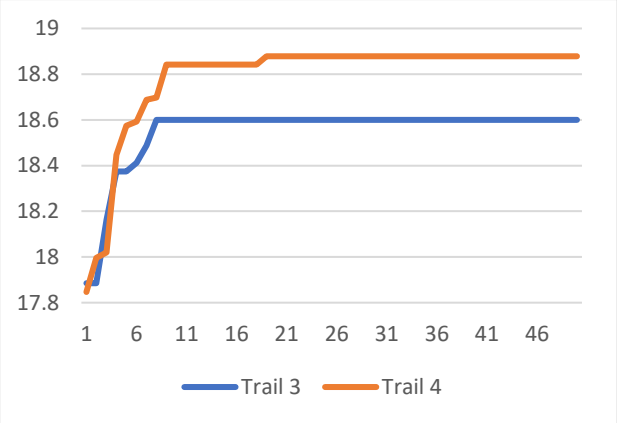


Figure 4 Best Fitness – Generation with learning rate

In trial 1 and trial 2, the fitness didn't change after 25 generations. With mutation method, the best fitness reached the best point in the 10th generation. Obviously, the mutation method has a great impact on the algorithm that it would help the algorithm converge more rapidly. What is more, the final average and best fitness in trial 3 are both 18.878 and they are both 18.599 in trial 4 which means that we could get a better fitness with a mutation. Therefore, the results implied that the mutation let the algorithm breakthrough the best fitness and make it possible for the algorithm to avoid keeping trap in a locally optimal solution.