Investigating parameters of genetic algorithm and neural network on classic snake game

Moo Chi Yuen
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP054804@mail.apu.edu.my

Lee Wuan Yeong
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP056174@mail.apu.edu.my

Edmund Chen Yi Kang
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP054563@mail.apu.edu.my

Shaheer Qaisar Syed
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP054386@mail.apu.edu.my

Zailan Arabee Abdul Salam
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
zailan@apu.edu.my

Abstract*— Genetic Algorithm (GA) and Neural Network (NN) are implemented in a classic Snake game to produce an optimal snake that can achieve high scores. GA serves as a snake evolution operator, whereas NN serves as a moving direction determinator. In this paper, several crucial parameters of GA and NN are modified to observe the influence of each parameter to the Snake game. GA has population and mutation rate regarding to the modifiable parameters, meanwhile NN has hidden layer and hidden neuron. The experimental results show how the snake game performs after modifying the parameters.Key words- Naive Bayes algorithm, text classification, Multinomial Naive Bayes (MNB)

Keywords*— Genetic Algorithm (GA), Neural Network (NN), variable parameters

I. INTRODUCTION

In this study, we decided to do research on Neural Network (NN) and Genetic Algorithm (GA) by looking the implementation on a very simply problem- The Snake Game. The snake game has basically 3 rules to survive which are eat the food, never hit the wall and never hit itself. Sharma [1] discussed about Genetic Algorithm that it is a natural evolution approach which get the idea come from Charles Darwin's principle. It includes selection, crossover and mutation. Karsoliya [2] discussed about Neural Network that it is built by mimicking the function of a human brain. It is formed by many neurons, and each neurons are assigned by weights to perform learning purpose. Hence, NN is good at local search, GA is good at global search. By combining it, in this case we have hybrid neural network with genetic algorithm which is being used in this study.

Hassanat, Almohammadi, Alkafaween, Abunawas, Hammouri and Prasath [3] proposed a journal article that concludes what mutation rates do to genetic algorithm, the values we pass into the mutation rate will affects how the program runs. It also suggested mutation rate should be balanced out with other parameters to reach optimum solution. Some dynamic approaches were suggested also, like controlling the mutation rate and crossover rates throughout the generations to maximize the algorithm. Experiments are made to compare against dynamic and static parameters, the results show that dynamic parameters have a better performance.

Sarmady [4] proposed a journal article to investigate the parameters of Genetic Algorithm. In terms of the parameter population, population sizes of 20, 100 and 200 were used to do the tests to find a suitable population size for further experiments. There wasn't a specific method for choosing the population size in the tests.

The size of 20 was considered too small as it wasn't enough to get a good enough result. The size of 100 and 200 gave similar results but the latter took double the time without a significant difference therefore size of 100 was considered as the suitable for further experiments. For the other experiments, the other parameters were changed but population size of 100 was kept constant throughout. Therefore, population size of 100 was the optimal solution for the project. If different sizes were picked, the results could have been different.

Karsoliya [2] discussed about approaches to determine the number of hidden layers and hidden neurons in neural network. First, rule-of-thumb method is discussed. It includes three rules:

- Number of hidden layer neurons < twice of the number of neurons in input layer
- Size of output/input layer < size of hidden layer neurons < size of input/output layer
- Number of hidden layer neurons = 2/3 size of input layer

Structured trial and error method was discussed for estimating number of hidden layers. It supposed that the researcher should increment the number of hidden layers and observe the results. Also, hidden neurons in the hidden layers should be equivalent. According to the rule-of-thumb methods, four hidden layers are not recommended because it

violates the rules, whereas one to three can be accepted. To conclude from the above, there is not an optimal value for number of hidden neurons and hidden layers that can satisfy any problems. Hence, researchers have to implement exhaustive methods which are time-consuming to find out the best solution based on the problems.

S. Panchal and Panchal [5] reviewed on methods of selecting number of hidden nodes in artificial neural network. There were two problems that need to put into consideration, the underfitting and overfitting. To find out the best result for choosing the numbers of hidden nodes, there were 5 methods that can be apply. The first method that can be apply is trial and error method. The method is run by repeating until the program reached its goal. Second, the rule of thumb method is discussed.

Third, the simple method, is by considering the input nodes, hidden nodes, and output nodes, with the same input output and output nodes, we can take the same number of hidden nodes. Forth, the two-phase method is most likely the trial and error method, but the data set was divided into 4 groups and by doing difference task to get the total number of error conditions for determining the number of nodes in the hidden layer. Lastly, the sequential orthogonal approach is by adding a neuron 1 by 1 until error is sufficiently small. In conclusion, the suitable number of hidden nodes can lead to a better result and less in time consuming.

II. Materials and Methods

A) Genetic Algorithm

For the genetic algorithm, the operators that used were the natural selection, fitness and the crossover and mutation. The natural selection was to create a population of 2000 snakes for each generation and to search for the best snakes to reproduce. Next, for the fitness, it was all about how the snake staying alive and getting a new high score. The strategy given for the snake to keep getting a new high score than rather staying alive without any improvement was to set a limit of 100 more moves, with a maximum of 500 moves.

In terms of crossover and mutation, the snake's brains were crossed with two different parents and developing a new child. The mutation happened after the process of the crossover and the brain of the snake will altered according to the mutation rate of 5%.

B) Neural Network

Each snake has a neural network to compute and determine the direction to move. As there are several types of Neural Networks available, a Feedforward Neural Network is implemented for this problem.

Fig. 1. shows the neural network implemented in the program, which has an input layer of 24 neurons, 2 hidden layers of 16 neurons, and 1 output layer of 4 neurons. The inputs are snake's vision in 8 directions searching for food, its own body and wall, whereby the outputs are the directions to move.

C) Computer Specification

- CPU: Intel® Core™ i5-8300H @ 2.30GHz, 4 Cores, 8 Threads
- RAM: 8GB
- OS: Windows 10 Pro

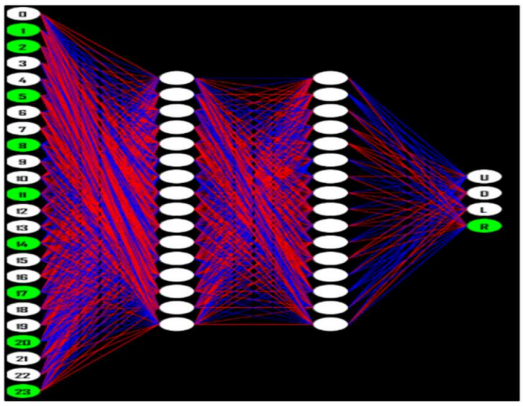

Fig. 1. Neural network of snake game

III. Result and Discussion

A) Mutation Rate

Mutation rate is the probability of mutation when it proceeds to the next generation, in this case, randomly swapping the bits of the chromosomes which represents the snakes. The snake program will be run for 3 times for each mutation rate. The mutation rates are 0, 0.05, and 0.8 in a constant population of 2000. To monitor the difference, a high score graph will be generated to display the output.

1) Mutation Rate 0.0

The mutation rate is set to 0.00 to study the situation when a genetic algorithm lost one of its key features which is mutation. Under this mutation rate, we can see that three out of three attempts having a very low score, and it does not increase when proceeding to the next generation as shown in Fig. 2.

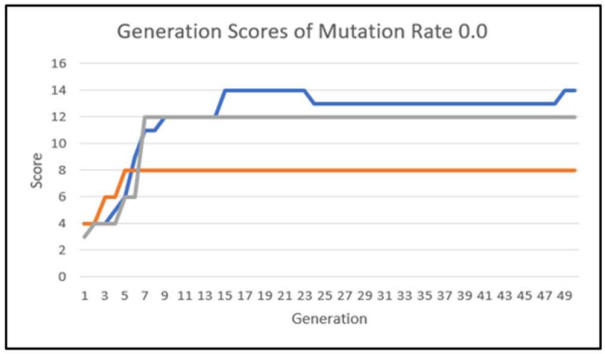

Fig. 2. Generation scores of mutation rate 0.0

2) Mutation Rate 0.05

This is a more common and efficient value as you can see the graph in Fig. 3. where the scores constantly went up across the generation. With this result, we can say that the "snake" is learning as the scores of it is constantly going up, in the very lately generations, it will reach its optimum solution.

3) Mutation Rate 0.8

Based on the graph in Fig. 4., the scores are growing in the early generations. However, in the late generations, the scores did not grow. It can say that it reaches its peak value but the

value isn't satisfied for the snake game as it will never finish the game, the peak value will be stuck for long generations. Although the score for it is high, but it only stops as it reaches its peak value.

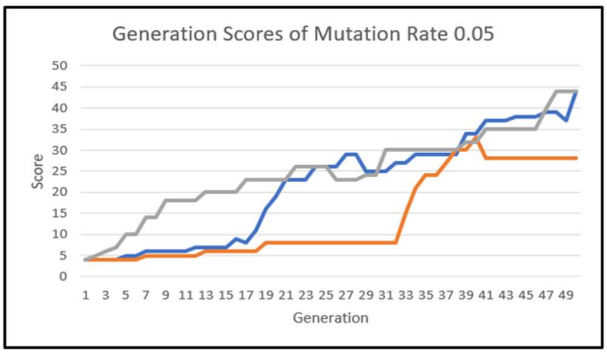

Fig. 3. Generation scores of mutation rate 0.05

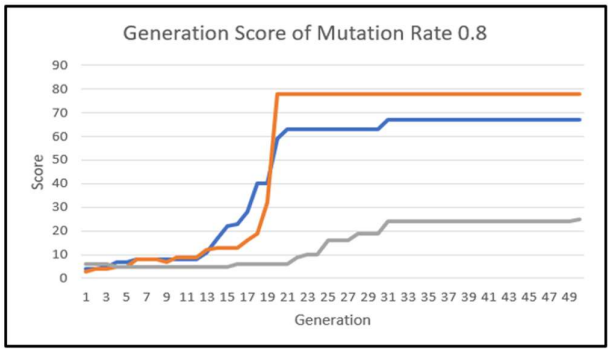

Fig. 4. Generation scores of mutation rate 0.8

The results of these observations can be explained as the mutation rate of 0.00 is too low and eventually out of the limit of the rate. The "snake" cannot learn anything without mutation rate, it is basically copying another "snake" with the same information, that is why the score is stuck and reaches its local maximum after a few early generations. However, for the mutation rate of 0.8 is too high for the "snake".

At early generations, the "snake" can learn but it doesn't perform stable. In late generations, it also reaches its local maximum and get stuck. This is because the mutation rate is too high for this case, the useful information in the generations does not get inherited most of the time, the search becomes too random. Lastly, mutation rate of 0.05 is more suitable for this population. The "snake" is learning and the score grows continuously.

However, 0.05 is not a magic number that is suitable for all different problems. The mutation rate has to be balanced with other parameters also, like population, crossover rates. To determine the mutation rate, experiments had to be made to test out which is more suitable for that problem.

B) POPULATION

Population refers to the number of snakes in each generation for this experiment. At the end of each generation, two parents will be picked based on the fitness score and selection method. Population sizes of 20, 200 and 20000 are implemented and each size was tested 3 times.

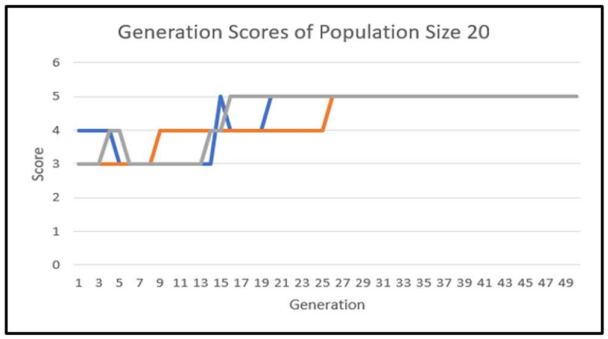

Fig. 5. Generation scores of population size 20

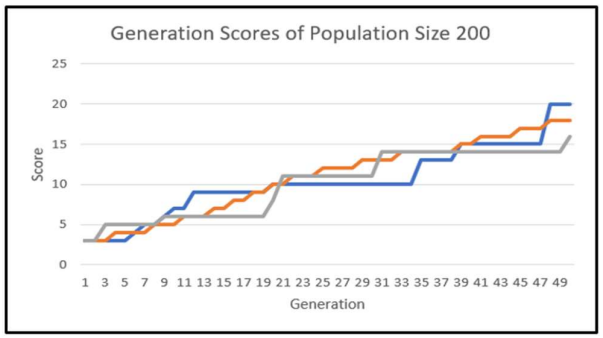

Fig. 6. Generation scores of population size 200

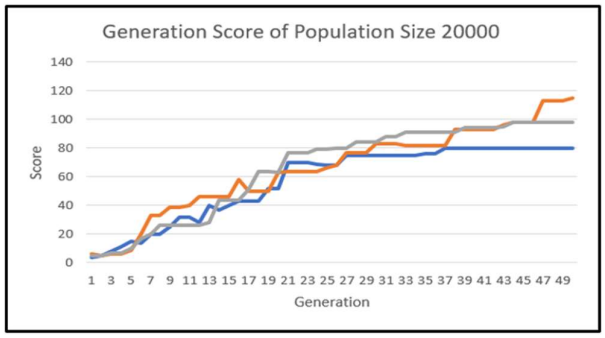

Fig. 7. Generation scores of population size 20000

The experiment results are shown in Fig. 5. – Fig. 7. When the population size was 20, although the time taken for the results was the quickest, the scores were very low compared to other population sizes.

When the population size was 200, the scores were only a little better compared to the previous size and the time taken was a bit more. When the population size was 20000, the scores were significantly better compared to the previous sizes even though it took a lot longer to get these scores.

Based on the observations, we can note that the population size is affecting the scores because the smaller population will have smaller chance to get 2 distinct snakes that have relatively high scores to be selected as parents for breeding purpose. According to this concept, the population size 20000 gave better results than other sizes, including the default population size of 2000.

C) HIDDEN LAYER

TABLE I. ENT RESULTS OF DIFFERENT NUMBER OF HIDDEN LAYERS

Hidden Layer	Experiment Results	
	Mean Score	Mean Run Time (mins)
1	97.8	137.0
2	66.4	93.4
3	62.8	126.6

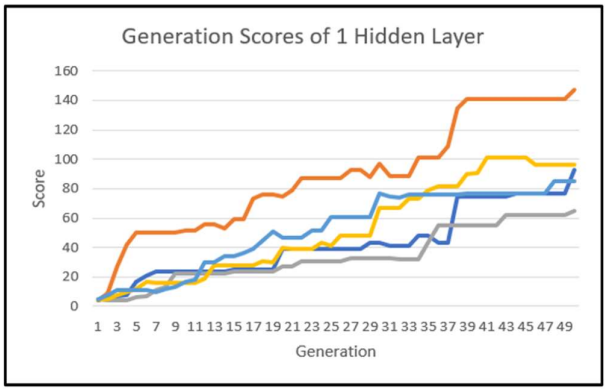

Fig. 8. Generation scores of 1 hidden layer

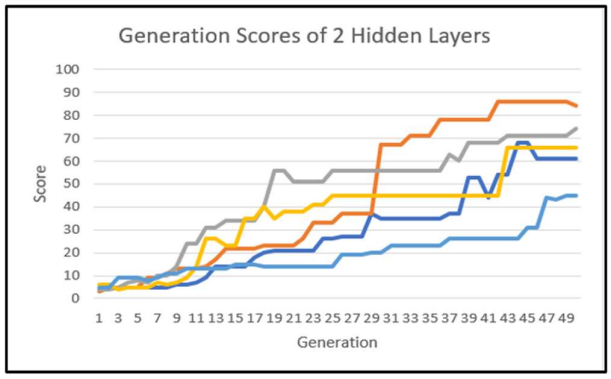

Fig. 9. Generation scores of 2 hidden layers

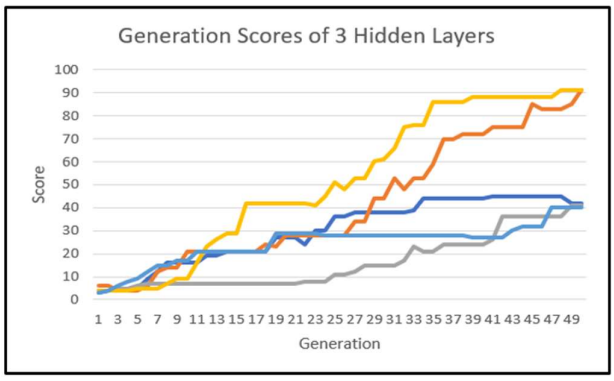

Fig. 10. Generation scores of 3 hidden layers

Fig. 8. – Fig. 10. show the experiment results of each number of hidden layers with 16 neurons. Each case has 5 times of execution. Table I shows the mean score and run time of each number of hidden layers with 16 neurons.

It is hard to evaluate the efficiency of each number of hidden layers because run time is proportional to the mean score, except when the snake can already achieve high score in the early generations. In terms of mean high score, it shows that one hidden layer performs better compared to two and three hidden layers as it has higher mean high score.

On the other hand, we observed that the movement of snake will become slower if more hidden layers are added. This is because it requires more computational time due to the complexity of neural network.

If each case has the exact same score trend, the highest number of hidden layers will have the longest run time. Hence, we can conclude that 1 hidden layer is the optimal choice for choosing the number of hidden layers.

D) HIDDEN NEURON

There are two specific result that have arisen due to the hidden nodes, the overfitting and underfitting. The program will be run 5 times by using the number of hidden nodes of 8, 16 and 22.

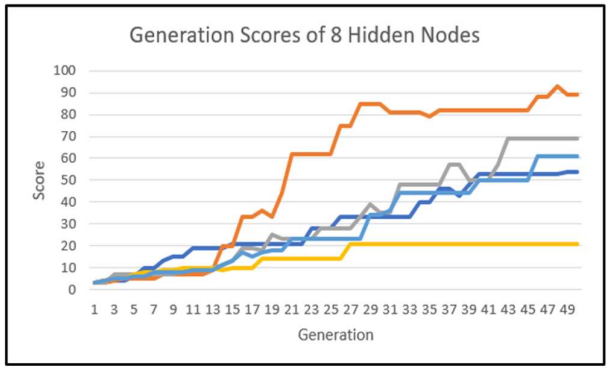

Fig. 11. Generation Scores of 8 Hidden Nodes

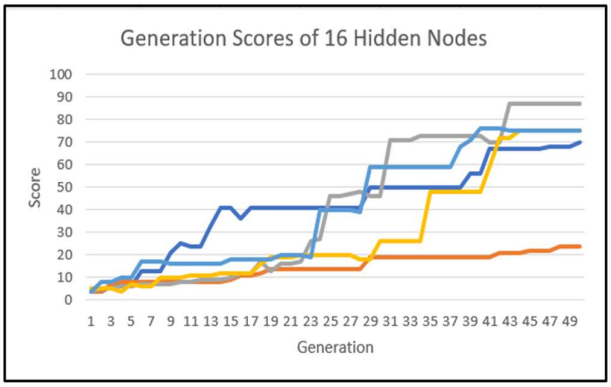

Fig. 12. Generation scores of 16 hidden nodes

Fig. 11. has a mean high score of 58.8, Fig. 12. has a mean high score of 65.8 and Fig. 13. has a mean high score of 46.4. By comparing the mean high score of Fig. 11. – Fig. 13., the highest mean high score belongs to the number of 16 hidden neurons, whereas the lowest belongs to the number of 22 hidden neurons.

In conclusion, we found out that the number of 8 hidden nodes is underfitting for solving the snake game while the number of 22 hidden nodes is overfitting and the most time consuming. Therefore, the number of 16 hidden nodes is the optimal choice.

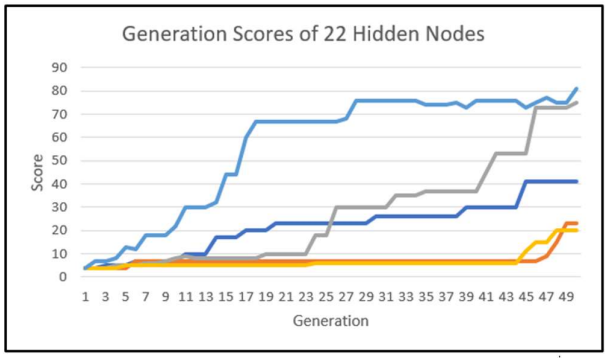

Fig. 13. Generation scores of 22 hidden nodes

IV. CONCLUSION

The experiments done were related to Genetic Algorithm and Neural Network which was implemented on a snake game. The findings of the experiments are specific to four parameters and each were tested individually without changing any other parameters. It is possible that if further experiments are done by changing the parameters together, better results could be achieved. The first experiment was done on the mutation rate and the best result was found when the mutation rate was set to 0.05. The second experiment was done on population size and the best result was found when the population was set to 20000. The third experiment was done on hidden layer and the best result was found when the hidden layer was set to 1. The last experiment was done on hidden neurons and the best result was found when the hidden neurons were set to 16.

REFERENCES

[1] M. Sharma, "Role and working of Genetic Algorithm in Computer Science role and working of Genetic Algorithm in Computer Science", International Journal of Computer Applications & Information Technology, 2(1), pp. 27–32, 2013.

[2] S. Karsoliya, "Approximating number of hidden layer neurons in Multiple Hidden Layer BPNN Architecture", International Journal of Engineering Trends and Technology, 3(6), pp. 714–717, 2012.

[3] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri and V. B. S. Prasath, "Choosing mutation and crossover ratios for Genetic Algorithms-a review with a new dynamic approach", Information, 10(12), 2019.

[4] S. Sarmady, "An Investigation on Genetic Algorithm Parameters", 2007.

[5] F. S. Panchal and M. Panchal, "Review on methods of selecting number of hidden nodes in Artificial Neural Network", International Journal of Computer Science and Mobile Computing, 3(11), pp.455–464, 2014.