AI Final Project Report CPSC-7373

Sai Ram Manohar Koya

T00708651

Title: Snake game automation using Genetic algorithm

1. Introduction

Genetic algorithm is a natural selection algorithm inspired by living creatures in nature. Genetic algorithm states that any problem can be solved by using three common techniques, *Selecion*, *Mutation* and *Crossover*. The Genetic Algorithm originated from a famous evolution theory , Darwin's theory of evolution, In this theory of evolution Davin states an interesting quote, '*survival of the fittest*', which implies that only those creatures who know how to survive. This is the major idea behind Genetic Algorithm development.

2. Relation between Genetic algorithm and Snake game

Genetic Algorithms are mostly related with genes and living creatures due to high relevance to the learning procedure of nature and living creatures over the period (ages). Most of the game cracks are about Q-Learning these days. A naive thought generated in us after listening to the performance and evolution of Genetic Algorithm is to test it with some real world examples. Relating a snake as a real creature and a box as an environment, we want to apply this genetic algorithm over this game to check if it works in any similar kind of situation.

Any creature needs a brain to process some query. In this case we assign a Neural Network as the brain of our snake to process the requests made by snakes in each state. The output of every state is in the form of direction, which implies the snake's direction to move. Here genes are weights and biases of neural networks, these are the key functioning units or decision modifying units of the neural network.

3. Abstract

In genetic algorithms, initially we have a huge lot of population created randomly without any rules and restrictions. We parallely release this population into the atmosphere or environment in which they are destined to live. After a certain time some of the population will die and some will survive despite any training, which means these kinds of creatures are what we require in future, because these will fit for our environment. Genetic Algorithm will Select the top fractional percent

of the population in order to produce the population for the next iteration. The steps performed on these selected populations to generate the next population are Crossover and Mutation.

4. **Preliminary approach**

We use just a feed forward neural network with our back propagation algorithm. Our requirement of a neural network is just to process the requests to see how far this network (or) brain of the snake can take it until it dies. So, a feed forward neural network will be sufficient for our present requirement.

We are not going to modify the weights or biases of the network through back propagation, rather we just want to remove this network or move forward this network to the next generation by evaluating its fitness score. In fact we just create a random number of neural networks of the same shape (we create a multiple number of weights and biases) and process all these parallely, In this context each neural network represents a different snake.

5. Key Words

Crossover, Mutation, Selection, Generation, Fitness score, Mutation rate, Crossover rate, Population size, Generation size

- Generation: current population is called generation.
- Population size is 1000 for our algorithm in every generation.
- Generation size is 100 (expected) for our algorithm.
- Mutation Rate : It is the fraction of the population to be mutated.
- Crossover rate : It is the fraction of population to be generated for the next generation through crossover of 2 selected parents from the current generation.
- Fitness score : It is a score assigned to a particular snake for evaluation of its survival.

6. Procedure

- Generation Growth

  Generation growth starts from an initial generation where we create a 1000 random snakes (neural networks of the same shape but different weights and biases). We select the parents first for crossover.

- Parent selection

  will select almost crossover_rate * population_size = 300 number of parents from the 1000 snakes. This selection is done by randomly selecting 3 snakes and selecting the winner of the tournament between these 3 initially selected snakes for 300 times.

ParentSelect Algo :

```
Parents = []
For i in 300:
     I = rand(), J = rand(), K = rand().
     Parents.push_back( winner( neuralNet[I],
neuralNet[J], neuralNet[K] ) )
Return Parents.
```

7. Crossover

The next part is Crossovering over the random pairs of parents to generate a new population. As we have to generate the crossover_rate * population_size = 300 number of children, we will apply crossover algo over 300 random pairs of selected parents.

```
CrossOver (Parent_1, Parent_2, crossover[] ):
     P1 = Parent_1, P2 = Parent_2
     S = RandomSelect( weight_or_neuron_or_Layer )
swap( P1(S), P2(S) )
crossover.push_back(P1, P2)
Return
```

8. Mutation

The next step is to Mutate the random previous generation neuralNets (snakes) and add Crossover Children, Mutated Children and to previous generation snakes.

```
Mutation Algo ( snake ):
     Temp_snake = snake
     S = RandomSelect( weight_or_neuron_or_Layer  )
     R = CreateRand( S )
     Temp_snake(S) = R
     Return Temp_snake.
```

Now in our neuralNet Array (or snake array) we have 2000 snakes, But we just need 1000 snakes to further process our next generation, So we run all these snakes parallely to determine their fitness score and we sort these snakes in the order of fitness score. Then we just chop off the last 1000 snakes in the sorted array to obtain the second generation snakes.

Then we repeat all the steps from beginning with ith generation snake neuralNets to get (i + 1) th generation snake neuralNets.

We define fitness of a snake at a point as the square of its length multiplied by its age.

```
F(x) = ( len(snake)  *  len(snake) ) *  age(snake)
```

Note : Further mutation of some random snakes are done after generation formation to get improved results.

9. **Storing and Loading**:

Weights and biases of top scored snakes in each generation will be stored in .npy files in the same directory as these weights and biases are numpy arrays in format. .nyp files loading and storing will be very fast when compared to simple I/O on text files.

These files will be loaded whenever needed for reference in order to process and test the behavior of these snakes. Of course the top generation snakes will be having the most probable top scores.

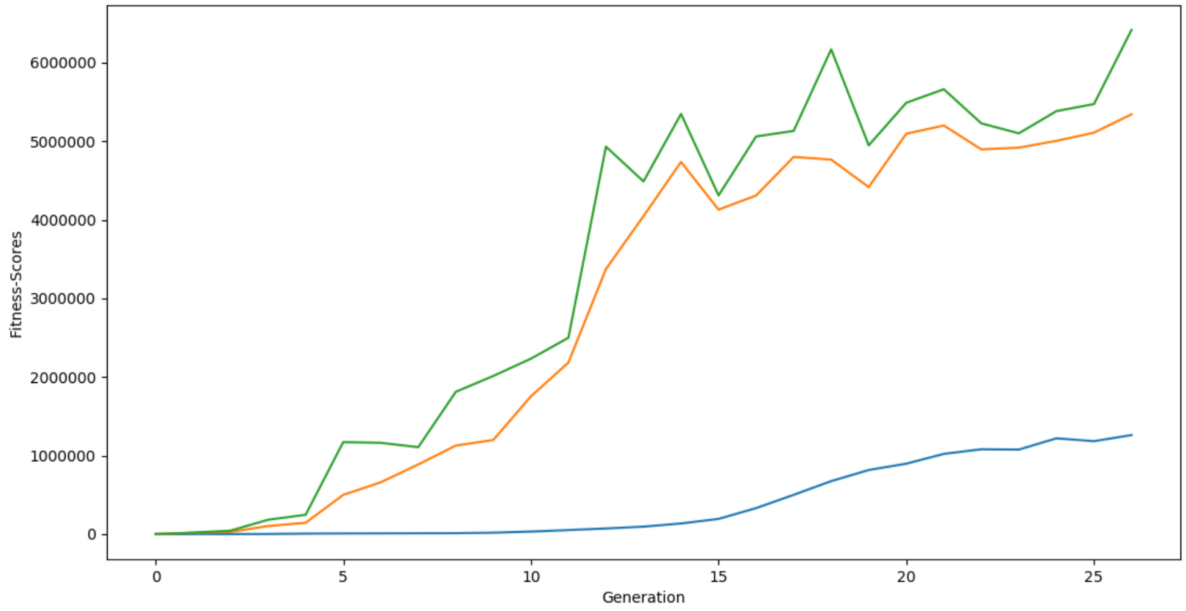
10. Hardware and software requirements

- Operating system: Any OS which can run Python v3.9 (Ubuntu, MacOS, Windows)
- Python libraries: numpy, numba, pygame, joblib
- RAM: minimum of 8 GB
- CPU: 3.2 GHz multi core processor
- GPU: not required, better to have for training performance improvement

All the above requirements are optimal and basic configuration to execute the project seamlessly. If the configuration used is better or worse than specified, there won't be any hindrance in statistics of results but there can be difference in the training time for computing weights and biases.

[Results in the next page]

11. Results



- Top scored snake fitness scores
- Top 6 snake’s average fitness scores in each generation
- Bottom 6 snake's average fitness scores in each generation