Implementation of artificial intelligence in Snake game using genetic algorithm and neural networks

Piotr Białas
Faculty of Applied Mathematics
Silesian University of Technology
Kaszubska 23, Gliwice, 44-100, Poland
email: piotbia887@student.polsl.pl

I. Introduction

Artificial Intelligence is recently growing in many areas. We see various developments for technical systems, image processing and multimedia solutions. We can find applications of intelligent technologies for smart management of energetic systems [1]–[3] and simulation models in thermal sciences [4], [5]. Also in last years gaming industry is using various aspects of intelligent technologies. There are many interesting system models which support interactions in real time [6]. Also computer systems support interesting application of gamification ideas [7].

Among techniques of artificial intelligence very high impact is visible from evolutionary methods. Algorithms are based on various models of life from animals [8], insects [9] or other. Systems which use various models seen in nature are used in image processing to track points of interest over images [10]. Similarly there are developed many hybrid solutions composed of neural networks working together with evolutionary one, like simulation methodologies for dynamic systems positioning [11] or smart control in homes [12]. Among those methods very interesting is genetic algorithm and its various versions. One of first propositions was presented in in [13] and it improvement was done in [14].

This article is about artificial intelligence implementation in the cult game - "Snake". How did I implement neural network (NN) and genetic algorithm (GA)? Which type of selection type I am using, the fitness function that helps me choose the best performing snake, parameters that are recommended to learn the fastest way e.g. size of the population, mutation rate etc. What are the inputs for NN and what are the outputs. To conclude I will talk about some statistics that I have prepared on different variations of GA parameters.

II. What is genetic agorithm

Genetic algorithm is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. John Holland introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution; afterwards, his student David E. Goldberg extended GA in 1989.
Source: https://en.wikipedia.org

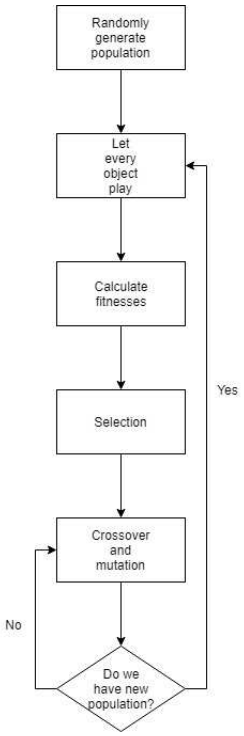
III. How does genetic algorithm work

First of all, we have to describe the snake implementation. It must have some neural network structure that will decide what action to take from any given input. This NN is called DNA in my project. This class has matrixes with weights and a separate ones with bias, which represent each layer of NN. There are methods responsible for computing output signal, mutation and crossover. These last two will be described later in the script. DNA class is the most important part of the snake – it is its "brain" that makes every decision. The next step is creating a function that allows us to calculate its performance. The performance should include the number of moves the snake executed without dying and scores.

- For the first step we need to randomly generate the population of objects, in our case the population of snakes. Randomly means, that his DNA's matrixes are filled with random float values between -1 and 1. In my case optimum size is 2000 snakes in population.
- Next, we let them play one after another. We start with the first one and after he dies, we start with another one repeating it until the last one dies. So now we know how many steps were executed and how many apples he ate.
- Now we calculate the "fitness" of each snake. It helps us see which one performed the best and which one should have the higher probability of being chosen for breeding. How the fitness is calculated is described in another section.
- Selection. Here we choose which pair of snakes – "parents" – will give their DNA to the new snake – "child". The probability of being chosen is based on fitness. The bigger it is the bigger the chances are. When we choose parents, we crossover their DNA – it means we take some of the weights from the father and some from the mother and apply it to their child. Again, selection types are described in another section. We repeat this process as long as we gain a new population.
- Every new snake after inheriting DNA from his parents goes through the process of mutation. We set % chances

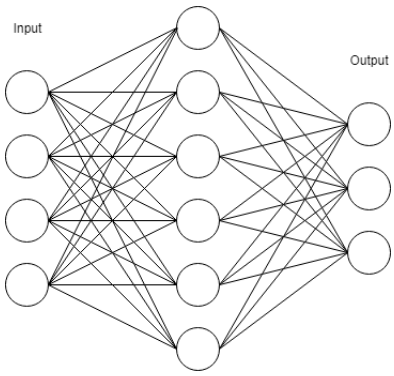
of mutation occurring – it is called “mutation rate”. Each value (weight and bias) have the same chance to mutate. If some weight mutates it changes its value to random value between -1 and 1. There are few different types of mutation that are covered later.

- We repeat steps 2 to 5 as long as we get the result we desire.

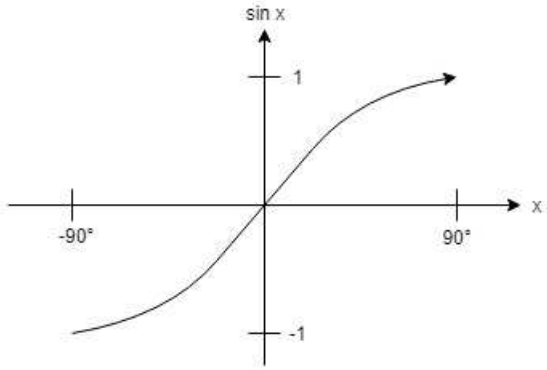


IV. Neural network structure, its inputs and outputs

My program is built in this way that we can choose how many hidden layers we want and how many hidden nodes will be in each one. My recommendations are 1 hidden layer with 6 neurons.



As for the inputs – there are 4:
- if there is a body part or wall on the left side of the snake’s head (0 or 1 value)
- if there is body part or wall in front of the snake (0 or 1 value)
- if there is a body part or wall on the right side of the snake’s head (0 or 1 value)
- a value of sinus of angle on which the food is inclined relative to the snake



For every input I include the direction the snake is moving in.
Why I used the sinus value? Because in range the from -90° and 90° it contains values between -1 and 1, so it demonstrates in which direction and how strong left or right the apple is. For example values (0, 1> insinuates that the apple is on right side, if it is closer to 0, food is more straight in front, if it is closer to 1, it means that it is much more at the right side of the snake.
There are 3 possible outputs. They indicate what will be snake’s next move.
- turn left
- go forward
- turn right

The snake’s brain chooses the most active one. And again, as for the inputs, they include current direction, so if we move South, the action are: left – East, forward – South, right – West.

V. Types of crossovers

A. One-point crossover

It treats matrixes as one straight line. We split each matrix of weights and bias in one point. The first half will be inherited from the father and second one from the mother. Variations of this method are choosing split points randomly or it can be fixed in code.

Father	0,35	-0,56	0,3	0,84	0,9	-0,15	-0,67	-0,14	0,01
Mother	0,25	0,6	-0,6	-0,47	0,54	-0,09	0,74	-0,23	0,61
Child	0,35	-0,56	0,3	0,84	0,9	-0,09	0,74	-0,23	0,61

B. Two-point crossover

The same case as in one-point crossover, but here we have a range from 2 to how many splitting points we desire. The child inherits values for change from mother and father. Also its location can be randomly chosen for every iteration or can be set in code.

Father	0,35	-0,56	0,3	0,84	0,9	-0,15	-0,67	-0,14	0,01
Mother	0,25	0,6	-0,6	-0,47	0,54	-0,09	0,74	-0,23	0,61
Child	0,35	-0,56	-0,6	-0,47	0,54	-0,15	-0,67	-0,14	0,01

C. Uniform crossover

We treat each value from matrix as individual. Then we draw a numbers between 0 and 1 and decide from which parent we inherit the weight. For example, if we obtain a number and it is lower than 0.5, we will take it from the father, otherwise we take it from the mother. The deciding value does not need to be exactly at half but it is recommended to be.

Father	0,35	-0,56	0,3	0,84	0,9	-0,15	-0,67	-0,14	0,01
Mother	0,25	0,6	-0,6	-0,47	0,54	-0,09	0,74	-0,23	0,61
Child	0,35	-0,56	-0,6	0,84	0,54	-0,15	0,74	-0,23	0,01

D. Arithmetic crossover

The final crossover that I will describe, is a mathematically calculated crossover. We take corresponding values from each parent and we take the average from it. The child's genes are these average values. There is a variation of this type. We can take random a percentage from one parent's weight and the remaining from other. For example 65% from the father and the remaining – 35% from mother. The proportion can be fixed or chosen randomly per each loop.

Father	0,35	-0,56	0,3	0,84	0,9	-0,15	-0,67	-0,14	0,01
Mother	0,25	0,6	-0,6	-0,47	0,54	-0,09	0,74	-0,23	0,61
Child	0,3	0,03	-0,15	0,19	0,72	-0,12	0,04	-0,18	0,3

E. My choice

In my crossover I chose one random number(a) in range <0, matrix's rows) and the second(b) from <0, matrix's columns). Using if statement as here:

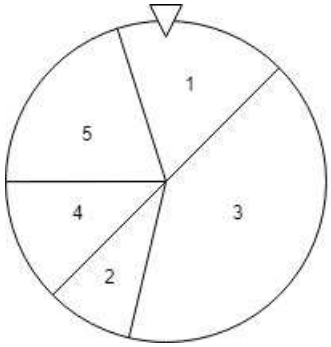
```
if ((j < a) || (j == a && k <= b))
```

Where j is current row and k is current column. If this condition is fulfilled the child inherits from the father, otherwise from the mother. In this scenario the matrixes are being cut from a curve so the crossover is very flexible.

VI. Types of selection

A. One-point roulette wheel

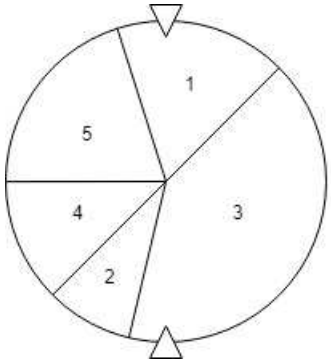
Each snake takes proportional (with its fitness score) space on a roulette wheel. We set fixed points on the roulette, so we know who will win. Then we spin the wheel twice. The 2 winners are chosen to be parents. The other modification is not spinning the wheel but randomly choosing the point. The chances in both scenarios are the same – the higher the fitness the higher the probability of being chosen.



Snake	Fitness	Chances
1	12	18.7%
2	3.5	5.5%
3	27	42.1%
4	7.2	11.2%
5	14.5	22.6%

B. Two-point roulette wheel

The same like in the one-point, but here we choose 2 fixed points opposite each other and when the wheel is spin, the 2 winners become parents. Or we can randomly choose 2 points and the fitness they land on will decide who the parents are. (Notice, that this and the previous method can be only applied for non-negative fitnesses).



Snake	Fitness	Chances
1	12	18.7%
2	3.5	5.5%
3	27	42.1%
4	7.2	11.2%
5	14.5	22.6%

C. Rank selection

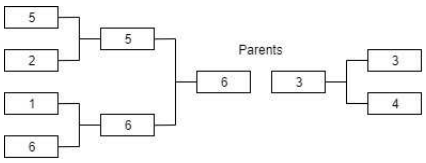
We sort snakes by the fittest one and give them ranks. The best one will get rank one, then rank two and so on. When it is complete, we perform the one-point roulette wheel selection, but based on ranks, not fitness. (The higher the rank, the higher the probability e.g. 1 – 40%, 2 – 30%, 3 – 20%, 4 – 10%). This method helps us pick the best performing snakes more often. We can have 3 snakes with fitness 21.8, 21 and 20.9. Based on their score they will have all similar chances to become

parents, but we require the best one (21.8) to reproduce, so we allocate ranks. (We can compare this to 40-meter runners where everybody has a very similar time, but still only the fastest one wins)

Snake	Fitness	Chances	Rank	New chances
1	22	22%	2	26.7%
2	19	19%	4	13.3%
3	23	23%	1	33.3%
4	16	16%	5	6.7%
5	20	20%	3	20%

D. Tournament selection

We choose a group of few snakes (from 2 to even half the population) and then we set brackets as in the football world championship. The best performing snake wins and proceeds the next round. In the end we have only 2 snakes left and they are the parents. This scenario always allows the best snake to reproduce because he will win in every group and with anybody, so the randomness is not so high and it can prevent the diversity.



Snake	Fitness	Chances
1	10	10%
2	12	12%
3	17	17%
4	9	9%
5	20	20%
6	32	32%

E. Random selection

This is the worst type of selection. We choose parents randomly without looking at their performance and fitness. In most cases it does not work.

F. My choice

In my selection I used the two-point roulette wheel, but firstly I sort snakes descending by fitness level. Then I set a variable called "Top" - decimal number between 0 and 1, which represents the percentage of top preforming snakes that will participate in roulette. Default value is 0.2 (20%). In that case I still have randomness throughout the population, but I do not allow weak snakes to reproduce.

VII. Mutation type

A. Whole mutation

Firstly, we create a variable and set its value between 0 and 1, which, again, represents percentage of probability of being mutated. When it comes to mutating, we reset weight value to a random number between -1 and 1.

Before	0,3	0,03	-0,15	0,19	0,72	-0,12	0,04	-0,18	0,3
After	0,3	0,03	-0,15	-0,85	0,72	-0,12	0,04	-0,18	0,3

B. Close mutation

The same as whole mutation, except this change the value randomly +-0.35, but not more than 1 and less than -1. The changing value can be randomly chosen.

C. My choice

I implemented the whole mutation method, because I belive it gives us more randomness and variation within the population.

VIII. Fitness function

A. Introduction

There are no rules that you need to follow during building fitness function. But you need to remember that the fitness should describe the best performing snake in the best possible way.

B. My choice

I used steps the snake executed and apples eaten. If the score is less than 10:

$$steps * steps * 2^{score}$$

And if it is larger:

$$steps * steps * 2^{score}$$

IX. Adjustable parameters

A. Hidden layers and hidden nodes

It is up to you how many of them you will have, you can have 0 hidden layers or 2 – it is up to you. 99 % of neural networks can solve your problem using 1 hidden layer at most.

B. Mutation rate

You need to adjust it by looking at results of your program. Mutation is important, because it helps algorithm to jump off of local minimum by keeping diversity. Recommended values are from 0.5

C. Population size

Firstly, you need to see how much can your computer run without lags. The size determines how diverse the population will be at the beginning. The size can be from 50 to even 10000 snakes.

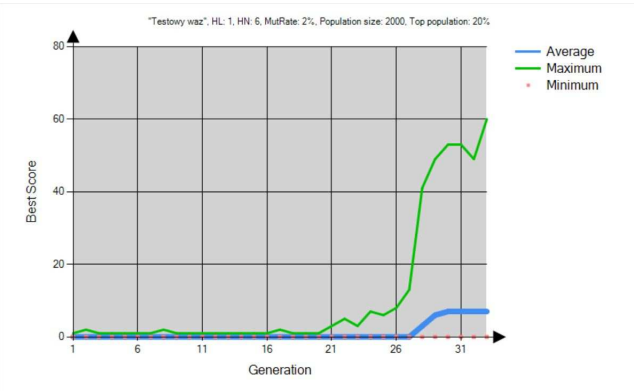
D. Top population

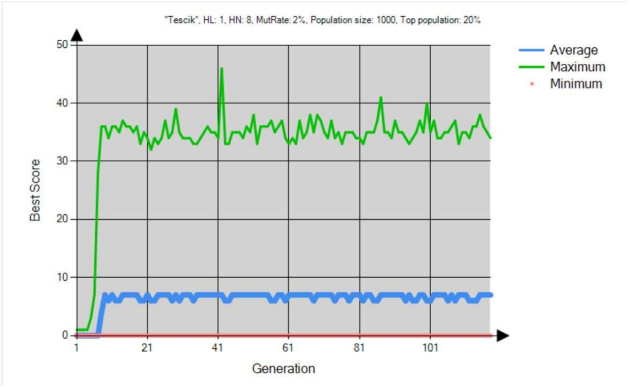
Variable, which describes how many percent of the best performing snakes will reproduce. I tried values between 5

E. Moves at start and moves for food

How long snake will live without eating and how many additional steps he will get from apple. I gave him 100 at start and 100 every apple he ate.

X. Statistics





Plots' author: Tomasz Siemek

XI. Conclusion

You do not need to use the same methods as I did. The best part of genetic algorithm is that you find by yourself which methods and which parameters are the best in your case. The NN structure can be different (e.g. more inputs, other inputs). In this article I described some of types and parameters, there are many more and you can even develop your own selection or mutation type.

References

[1] G. Capizzi, G. L. Sciuto, C. Napoli, and E. Tramontana, "Advanced and adaptive dispatch for smart grids by means of predictive models," *IEEE Transactions on Smart Grid*, vol. 9, no. 6, pp. 6684–6691, 2017.

[2] F. Bonanno, G. Capizzi, A. Gagliano, and C. Napoli, "Optimal management of various renewable energy sources by a new forecasting method," in *International Symposium on Power Electronics Power Electronics, Electrical Drives, Automation and Motion*. IEEE, 2012, pp. 934–940.

[3] F. Bonanno, G. Capizzi, S. Coco, A. Laudani, and G. Lo Sciuto, "A coupled design optimization methodology for li-ion batteries in electric vehicle applications based on fem and neural networks," in *2014 International Symposium on Power Electronics, Electrical Drives, Automation and Motion*. IEEE, 2014, pp. 146–153.

[4] G. Capizzi, G. L. Sciuto, G. Cammarata, and M. Cammarata, "Thermal transients simulations of a building by a dynamic model based on thermal-electrical analogy: Evaluation and implementation issue," *Applied energy*, vol. 199, pp. 323–334, 2017.

[5] R. Brociek and D. Słota, "A method for solving the time fractional heat conduction inverse problem based on ant colony optimization and artificial bee colony algorithms," in *International Conference on Information and Software Technologies*. Springer, 2017, pp. 351–361.

[6] I. Martišius and R. Damaševičius, "A prototype ssvep based real time bci gaming system," *Computational intelligence and neuroscience*, vol. 2016, p. 18, 2016.

[7] D. Ašeriškis and R. Damaševičius, "Gamification patterns for gamification applications," *Procedia Computer Science*, vol. 39, pp. 83–90, 2014.

[8] D. Połap *et al.*, "Polar bear optimization algorithm: Meta-heuristic with fast population movement and dynamic birth and death mechanism," *Symmetry*, vol. 9, no. 10, p. 203, 2017.

[9] R. Brociek and D. Słota, "Application and comparison of intelligent algorithms to solve the fractional heat conduction inverse problem," *Information Technology and Control*, vol. 45, no. 2, pp. 184–194, 2016.

[10] M. Woźniak and D. Połap, "Bio-inspired methods modeled for respiratory disease detection from medical images," *Swarm and evolutionary computation*, vol. 41, pp. 69–96, 2018.

[11] ——, "Hybrid neuro-heuristic methodology for simulation and control of dynamic systems over time interval," *Neural Networks*, vol. 93, pp. 45–56, 2017.

[12] ——, "Intelligent home systems for ubiquitous user support by using neural networks and rule based approach," *IEEE Transactions on Industrial Informatics*, 2019.

[13] J. rey Horn, N. Nafpliotis, and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence*, vol. 1. Citeseer, 1994, pp. 82–87.

[14] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *International conference on parallel problem solving from nature*. Springer, 2000, pp. 849–858.