Adaptation to environment using a genetic algorithm

Erik Asp[1], Julius Kördel[2]

Abstract
A neural network is a method for implementing an AI that uses experience and rewards to change and adapt so it can perform a task as good as possible. How the neural network changes is calculated using a genetic algorithm. This report explains a project that via an implementation of a neural network and genetic algorithm evolves a simple moving cube to hunt other cubes. The project uses various genetic algorithm techniques such as elitism, crossover and mutation to evolve the AI to perform as exceptional as possible. The result is an AI that evolves depending on starting state and the values of different parameters.

Authors
[1] *Media Technology Student at Linköping University, erias104@student.liu.se*
[2] *Media Technology Student at Linköping University, julko800@student.liu.se*

Keywords: Neural network — Genetic algorithm

Contents

1. Introduction

Artificial intelligence can take many shapes and forms. It can be anything from a guided system aimed to make someone perform better, a pattern recogniser/classifier to something that can be interacted with that act in an intelligent way.

A branch of AI are neural networks and the process of training the neural networks with genetic algorithms, which through emulating natural occurring phenoms strive to breed an AI that is the most fit for solving the task at hand.

This projects purpose is to create an AI that seeks out prey with a behavior obtained by using a neural network and a genetic algorithm. The AI or "Predator" takes different input variables into consideration when it decides how to act and depending on its performance it has a chance to breed the next generation. The "Predator" will be referred to as a seeker.

2. Theory

In this section the needed theory to understand the algorithms used in this report are presented.

2.1 Neural network

Neural network is a method for modeling how the human brain works. A simplistic way of describing the human decision making process is that when a human makes a decision, a nerve impulse traverse sequences/path of neurons and activates them. This process is what neural networks mimic [1].

When modeling neurons they are described as a core that has inputs and outputs. To model the strength of the nerve impulses this project uses weights, weights describe how strong the impulse is and directly affects the output of the neuron core.

Neurons connected to neurons can be structured into layers and there are 3 types of layers [2]. Input layer, output layer and hidden layer. The hidden layer is the one between the input and output layer. Each of these layers can have any number of neurons and the number of hidden layers can increase the performance but it comes at the cost of increasing complexity. An example of a neural network with all these

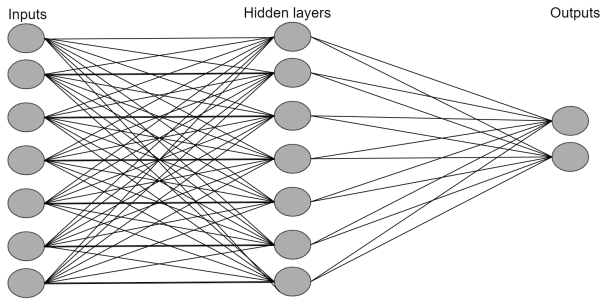


Figure 1. The used neural network with 1 input layer and 2 outputs

layers can be seen in Figure 1. The neural network itself is not intelligent from the start but through the process of training, weights can be decided that results in desired output.

2.2 Genetic algorithms

Genetic algorithms is a group of algorithms inspired by natural selection and evolution [3]. The idea is to optimize the neural network to make the best decisions by relaying performance when selecting who will breed for the next population combined with mutations in breeding and crossover between individuals that breeds.

Just like in nature there are different variants of natural selection and because of that there are different versions of genetic algorithms as well, two major versions are Elitism and Chromosome representation.

Elitism is based on that the most well adapted will survive and only the best are allowed to pass on their genes to the next generation. On the other hand, in Chromosome representation the genes are represented as a list and then the mutations/variations happens on the elements in the list. These can be implemented separately but they can also be mixed together.

3. Method

How the theory is implemented in this project is presented in this section.

3.1 Inputs for seeker

The seeking AI has 7 inputs that has been normalized to the range of 0 to 1. 5 of these inputs corresponds to what the AI can see in it's field of view that can be shown in Figure 2 where each white line corresponds to an input and has a max range of the seekers view radius. If the line does not hit anything its value is 1 and then becomes closer to 0 when the distance to the prey gets smaller. The distance is calculated by the length of the line that represents the field of view, an example of an interception of this line is shown in Figure 3. Further it also has 2 inputs about prey. First one is if the predator sees a prey, yes or no. The second one is how far away is the prey if i see it.

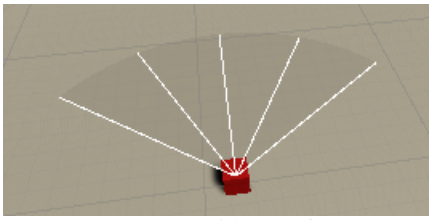


Figure 2. Seeker with rays for sensors

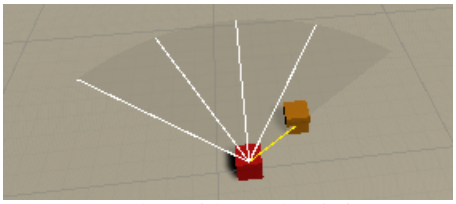


Figure 3. Seeker with ray that is intercepted

3.2 Neural network and genetic algorithm

To make an individuals neural network evolve, the elitism method that was mentioned in section 2.2 is used. In the case of a predator searching for prey in a static scene it is more efficient to use the elitism method. This because when the scene is static, the best individual can easier adapt to become the "perfect" individual. The next best individual was also used and together the two became "parents" to the next generation that was evolved by using crossover and mutation.

3.2.1 Crossover

One key aspect to make the individual evolve is crossover. The crossover method takes the best individuals, which in this case corresponds to the individuals that got the highest scores. The weights in the two best individuals networks is then used to create a new network using the combined weights of the first two and assign this new network to a new individual [4].

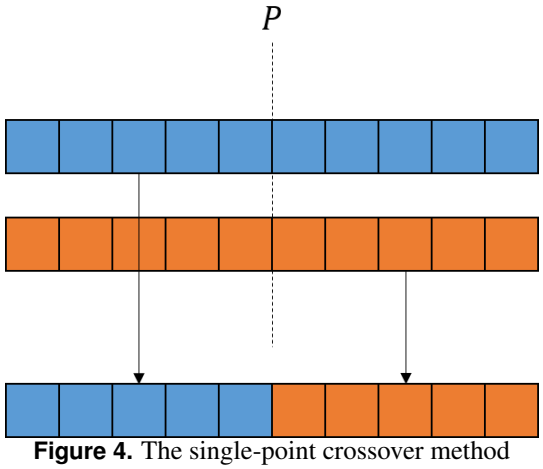


Figure 4. The single-point crossover method

In Figure 4 the single-point crossover method is illustrated. In single-point crossover a random point between 0 and the total number of weights is chosen. It is at this point where the networks are cut and the new network is built up using the weights from 0 to the cutting point of the first network and

the rest from the second network. Which of the networks that is first is also random.

3.2.2 Mutation

To ensure that the individuals evolves further to reach its maximum potential, mutation is used. Mutation takes the weights of the individuals neural network and changes them slightly to add some random behaviour so that changes to the neural network still can occur [4].

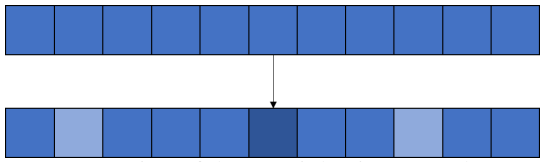


Figure 5. Mutation of three weights in a neural network

Figure 5 shows the mutation process. Three weights are changed to make the neural network evolve further. Mutation is applied after crossover.

3.3 Implementation

To implement the scenes, the game engine Unity was used. The code written to run the implementation was written in the language C#. At the time this was implemented there were no sufficient libraries in Unity to implement neural networks, therefore the neural network needed to be implemented from scratch.

3.3.1 Implementation of population

The implementation of the population that used the neural networks was made by creating a number of individuals and then use the ones that performed the best to evolve. The most frequent number of individuals was 16. The reason for this is that more individuals result in possibility for more different weights which increases performance, there were no more than 16 because the monitoring of the performances would have been more difficult. Instead of using crossover and mutation on all individuals, they are divided into four "selection groups" that evolve differently. The first four use random weights each generation of the evolution, the second four evolves using only crossover, the third four uses only mutation and the last four uses both crossover and mutation. All 16 scenes with individuals can be seen in Figure 6.

The reason why the different individuals evolves differently is so that the evolution will not get stuck in a local maximum. If the weights are close to the same on all individuals, it can get stuck in the same pattern over and over again without increasing the scores. The individuals with random weights is therefor useful to get a better score if the local maximum is a low score. Using only mutation can also keep the networks from getting worse the other way around by maintaining a high score if one has been achieved. The crossover and crossover with mutation are there to increase the performance of a network that has a high score but can get higher.

Figure 6. 16 individuals running simultaneously

3.3.2 Scores

To determine which individual that performed the best, a score system was necessary. The goal was to reward the individuals that caught prey as fast as possible and to punish those that just ran around and did not do anything.

$$score = \begin{cases} score - 60 \cdot t \\ score + 1000 + (2 \cdot t_{left}) \end{cases} \quad (1)$$

Eq. 1 shows how the score changes through time, *score* is the score of an individual, t is the time the individual spends not seeing a prey and t_{left} is the amount of time left before the genetic algorithm is triggered and the scenes reset. The top part is the reduction of the score when the predator can not see a prey and the bottom part is when the predator catches a prey. If the predator sees a prey the score remains unchanged.

The score system was designed this way so that the best individual was the one that had a predator which catches prey as fast as possible, that the best individual prioritize prey meaning it want to at least look at the prey and preferably move forwards them.

3.4 Parameter tuning

All parameters that has been varied over different runs can affect the result and a way to evaluate a genetic algorithms performance is needed to optimize it. Here is were the subject of parameter tuning and genetic algorithm analysis can be used in a systematized way [5], but for this project a a more exploratory approach was used.

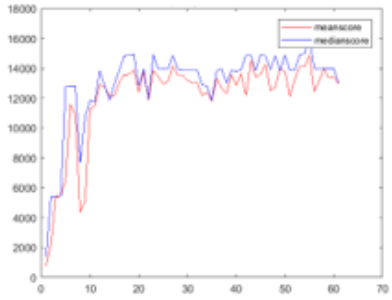
4. Result

There are a number of parameters in this implementation that can be modified and those are: Number of individuals per generation, prey per generation, number of obstacles, mutation rate, perpetuation rate and number of crossover points.

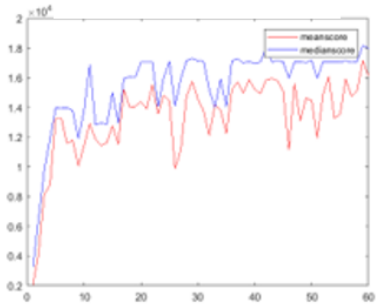
All runs in this section starts with random weights for all individual in the population. If nothing else is mentioned, presented results is the product of running for 60 generations. The performance for different runs is presented in table 1.

The first two entries of table 1 shows a performance increase if the number of individuals are increased. This was early noticed and thus the rest of the runs are performed with 16 individuals instead of 9. The table also shows from entries three and four that when using different breeding groups the result performance better in both cases compared to previous breeding method where only mutation of the best were used.

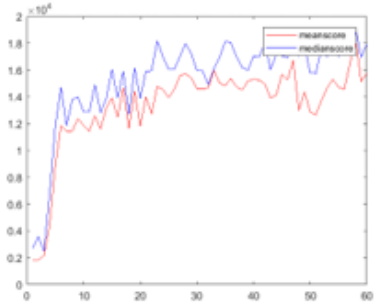
To display how parameters such as the mutation rate and perpetuation affected performance during presented runs, the parameters is plotted against generations to showcase the change over time. For mutation rates see figures 7 and 8. For perpetuation rates see figures 9 and 10. These rates were based on rates used in [6].



(a) 0.011 mutation rate

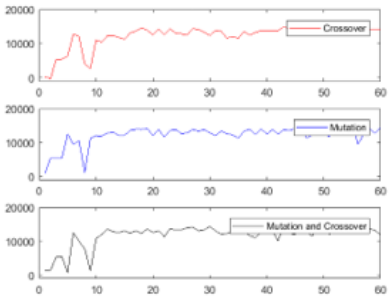


(b) 0.1 mutation rate

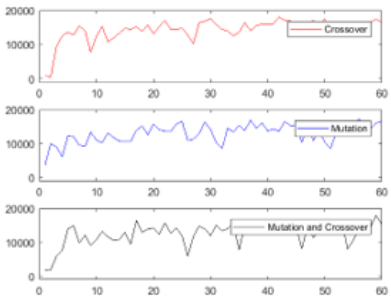


(c) 0.2 mutation rate

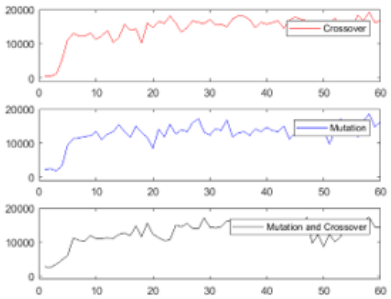
Figure 7. Mean and median for different mutation rates



(a) 0.011 mutation rate



(b) 0.1 mutation rate



(c) 0.2 mutation rate

Figure 8. Crossover, mutation and crossover+mutation for different mutation rate

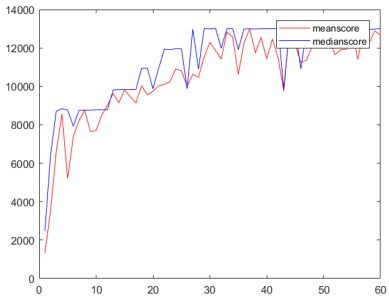
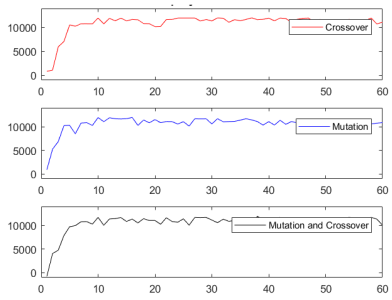
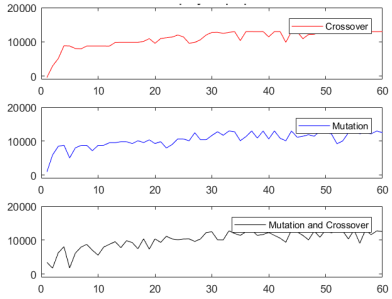


Figure 9. 0.6 perpetuation rate



(a) 0.3 perpetuation rate



(b) 0.6 perpetuation rate

Figure 10. Crossover, mutation and crossover+mutation for different perpetuation rate

The table 1 shows that mutation rate affects the end result but perpetuation itself does not. However, the table does not display how parameters affect the results during the runs. The difference between runs is shown by comparing the results of highest and lowest mutation rate entries five and seven in table 1 and the different perpetuation rates result in entries eight and nine in table 1. What graphs 7 and 8 showed is that higher mutation rate makes the result for each iteration deviate more from previous solution and that deviation can result in a better, but also worse result.

The results of keeping all settings except the perpetuation rate, that was doubled, shows that its affect is small. Worth mentioning is that even though the perpetuation rate was doubled, mutation rate was fairly low at a rate of 0.011, which might have affected the results.

Entries 14, 15 and 16 in table 1 displays how different training environment affects the results. The table shows that training with obstacles from the start is better than training without them if the goal is to perform on a scene with obstacles. The best result for a scene with obstacles was given by first training without and then continue training that AI with obstacles. The provided scenes for these entries are the same.

5. Discussion

In this section the findings of the result and methods used will be further discussed.

5.1 Faults with evaluation/comparisons of result

One of the biggest critiques for the implementation is the dependence on a good start. For example if all seekers gets poor weights at the beginning and all end up standing still, it will take longer for this scenario to reach a good performance. This is why always having a part of the generation being completely random was implemented. If there always is a part that is random the probability of breaking out of the current behaviour(if it has poor performance) increases.

The prior logic of random start combined with that many of the results runs has been on random generated scenes and not a static scene means that they might not be comparable. For a pure comparison of parameter against parameter everything else except what we are examining should be the same. However the reason for having a random scene (position of prey and possible obstacles) was that the bias to a specific scene would be minimized. This was important because it was longed for that the implementation would work on different environments(given that it had time to train on it).

5.2 Parameters for success

There are parameters that by them self increase the performance of the population and there are those that need to be combined to increase performance. Increasing the number of individuals per generation and having selection groups is of the first mentioned type. Mutation rate, perpetuation rate and crossover point is of the second type. Therefore if parameters are to be selected for success then more individuals are a must and preferably the individuals should be divided into different groups. For breeding the different rates depends on how they are combined, one combination is a high mutation rate in range of 0.1 to 0.2 with a lower perpetuation rate of 0.3 and another combination is a lower mutation rate in range 0.011 to 0.1 and a perpetuation rate of 0.6.

The number of prey for optimal success is a tricky question, because increasing the number of prey will also increase the number of prey the seeker must catch, their score might be higher but the catching percentage of prey has decreased. This leads to that the number of prey themselves is not a factor to easy derive to how it affects the result but a try and error method should be applied.

5.3 Nature of genetic algorithms

The genetic algorithm is good at learning the problem it was presented with, but it is easy to throw it off. For example if the seekers learn that there is a specific path/pattern that is optimal for the given layout of the scene. Then if for the same seeker there is a obstacle placed in the way, it will take time to adapt to this obstacle. This is also a good reason for including a random section to the population. Because if the environment is changed, then what previously was the best might be the new worst possible solution and thus a new random way of approaching the problem might be more successful.

If the implementation runs at a very long time the optimal behavior will converge, that means that the best and second best used to breed the next generation are so similar that next

generation becomes more and more equal and the probability of overfitting increases. Thus it might not be worth running the algorithm for too long, but there is always possible that the current solution is but a local maximum and thus if we run for a longer period of time, a greater solution might occur at the cost of run time and overfitting.

6. Conclusion

Genetic algorithms for learning the weights of neural networks can be very useful for creating an AI to solve specific type of problems, or problems of a similar setting. However it might not be the best solution for solving a general problem if the training has not been on a general problem basis, generally the more specific it is trained the better it will perform on that problem and the worse it will perform on other problems. It will sooner or later adapt to this new environment, though it might take longer than for a new random individual. Much like in real life.

References

[1] Wikipedia. Artificial neural network, October 20, 2019.

[2] ujjwalkarn. A quick introduction to neural networks, August 9, 2016.

[3] Geek for geeks. Genetic algorithms, No date.

[4] J.H. Holland. Adaptation in natural and artificial systems. *MIT Press*, pages 97–106, 109–111, 1992.

[5] A.E.Eiben and S.K.Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. page 20, 2011.

[6] J. E. ANGUS R. N. GREENWELL and M. FINCK. Optimal mutation probability for genetic algorithms. *Human Nature*, pages 8–11, 1995.

1. Tables

"No data" in any of the measures means that it was not recorded for that instance. The parameters of the table 1 are as follows:

- Individuals, the number of individuals that exist in given population.
- Prey, the number of prey existing at the start of the generation.
- Obstacles, things in the scene that makes navigating and catching prey harder.
- Mutation rate, how much the genes can change.
- Perpetuation, how many of the genes that has a possibility to mutate.
- Time limit, round time before one iteration ends.
- Average, the average score of all individuals over all iterations. The first four with random genes are excluded.
- Median, the median score of all individuals over all iterations. The first four with random genes are excluded.
- Mutation, the average score of all individuals with only mutation over all iterations.
- Crossover, the average score of all individuals with only Crossover over all iterations.
- Mutation + Crossover, the average score of all individuals with mutation and crossover over all iterations.
- Other, information that is specific for this run.

Table 1. Results for runs of different settings

	Individuals	Prey	Obstacles	Mutation rate	Perpetuation	Time limit	Average score	Median	Crossover	Mutation	Crossover + Mutation	Other
1.	9	7	0	0.01	0.3	30 s	1247.3	1411.9	No data	No data	No data	No selection groups
2.	16	7	0	0.01	0.3	30 s	3952.6	4458	No data	No data	No data	No selection groups
3.	16	7	0	0.01	0.3	30 s	No data	No data	4542.5	4910.8	4426.1	
4.	16	7	0	0.01	0.3	30 s	No data	No data	5481.8	5814.6	5530.1	
5.	16	21	0	0.011	0.3	30 s	12038	13030	12193	12213	11706	
6.	16	21	0	0.1	0.3	30 s	13350	15468	13016	14348	12685	
7.	16	21	0	0.2	0.3	30 s	13322	15286	12781	14514	12670	
8.	16	14	0	0.011	0.3	30 s	No data	No data	10700	10939	10554	
9.	16	14	0	0.011	0.6	45 s	10466	11292	10443	10901	10053	
10.	16	14	3	0.011	0.3	45 s	No data	No data	7734.6	8394.6	7552.3	
11.	16	14	3	0.011	0.6	45 s	13442	15169	12983	12832	14512	
12.	16	20	5	0.011	0.3	30 s	No data	No data	7734.6	8394.6	7225.3	
13.	16	20	5	0.011	0.3	30 s	8567.2	9850.8	8709.1	8635.1	8357.3	
14.	16	21	3	0.011	0.3	45 s	8221.1	9959	8315.5	8439.8	7907.9	
15.	16	21	3	0.011	0.3	30 s	13205	14689	13278	13599	12739	
16.	16	14	3	0.011	0.3	30 s	8029.9	9488.7	8163.6	8218.8	7707.4	Trained with obstacles
17.	16	21	3	0.011	0.3	30 s	3387.1	6630	7300.5	35	2825.8	Trained without obstacles, then used with obstacles
18.	16	21	3	0.011	0.3	30 s	13205	14689	13278	13599	12739	Trained with obstacles, then used without obstacles