Self driving AI - Neural Networks and Genetic Algorithms

Juan Pacheco Larrucea

July 2022

1 Introduction

Since cars were invented, the problem of creating an automatic driver started to rise. This problem has been tried to be solved in real life by many companies, but this is not the only place where this has been hanging around. In racing video games, the idea of having a good automatic driver as an enemy is something that has been explored through the years. Many valid solutions have been researched, but with the rising of machine learning the usage of these innovative mechanisms has been something of interest for game developers.

This research tries to explore how some of these mechanisms work and how can they be used to make an AI that learns how to drive on a map. The racers will follow an evolutionary system along the usage of Neural Networks to improve their racing skills.

2 Problem statement

The idea of this project is to make a car AI (that is always speeding up) learn how to drive in a circuit using a Neural Network and by making it learn from previous generations of cars that have failed in the same task, selecting the ones that performed better to improve.

Better solutions exist for this specific problem, but the idea of this research is to explore how evolution can modify a brain to educate it to make the correct guesses, and after that, if the method succeeds, it could be applied to other problems.

3 Circuit

In this framework, the representation of the circuit is done with simple lines and a circle, The white lines represent the circuit boundaries, if a car touches them it will crash and it will get deactivated. The green lines represent the interesting points of the circuit; they have a specific order, and when a car crosses them in the correct order the heuristic value of the car will increase [Code Bullet 00]. Finally, the circle represents the starting points and the blue line represents the starting direction of the cars.

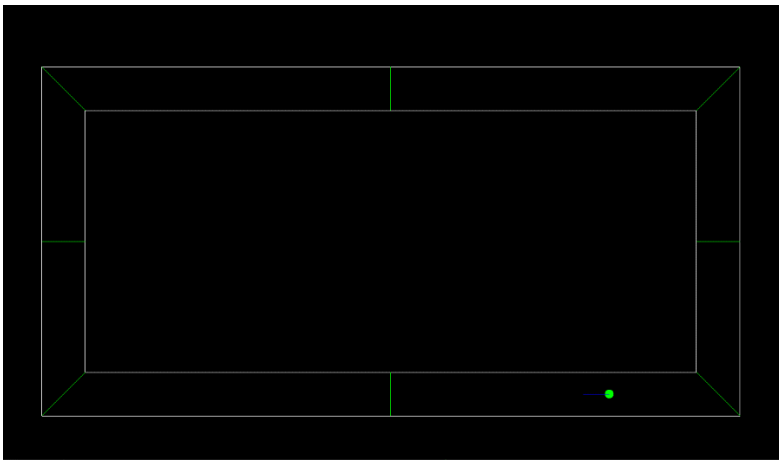

Figure 1: First Map.

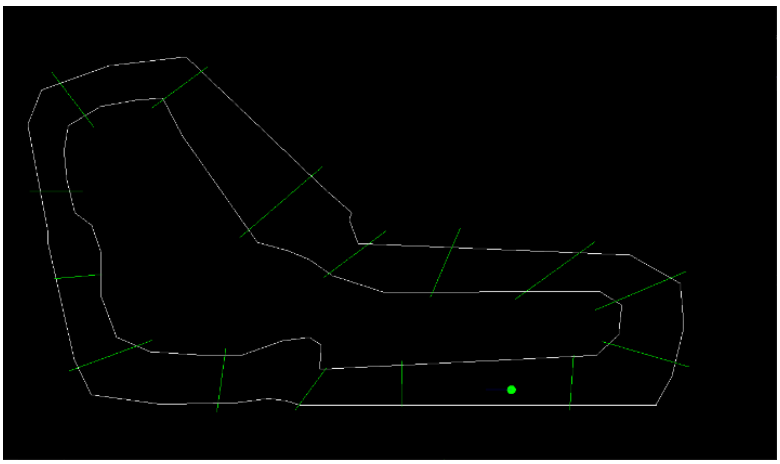

Figure 2: Second Map.

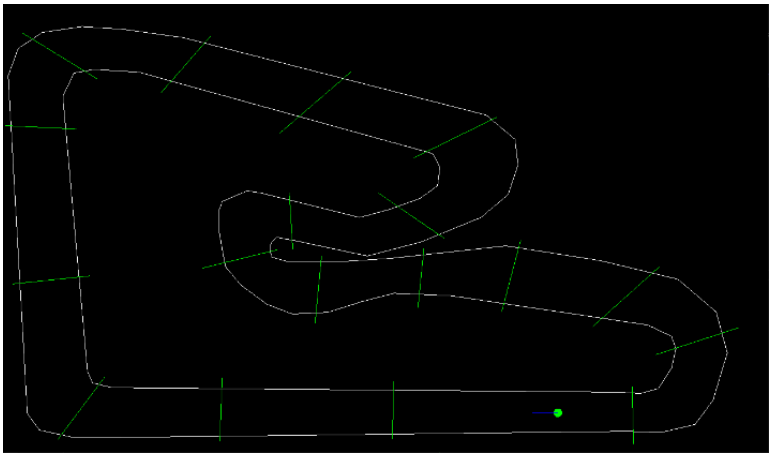

Figure 3: Third Map.

4 Cars

The cars are the main agents of this AI. They are represented by a white square and they are always speeding up, only being able to turn left or right.

4.1 Sensors

These cars need a way to sense the environment, so they have a specific number of rays (coming out of them) that are cast against the circuit boundaries to tell the car the distance to a wall. After some tests, I have found that an optimal number of rays without wasting too much performance and without making the AI not able to sense at all is 5, so I will stick to that number.

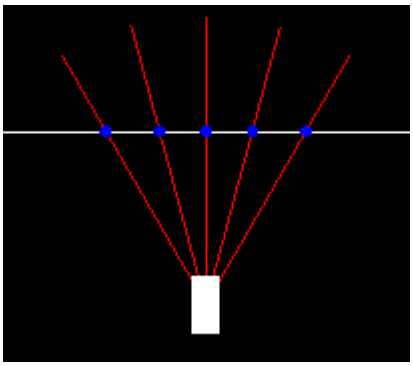

Figure 4: Car rays sensing the environment.

4.2 FOV

The rays of the cars are spread equally in an area that has the same direction the cars have. This FOV is modifiable and is useful for the car to have more control of the environment.

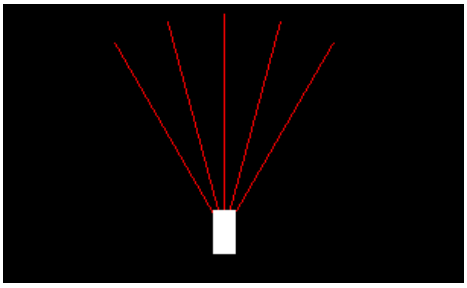

Figure 5: Car with FOV 60º.

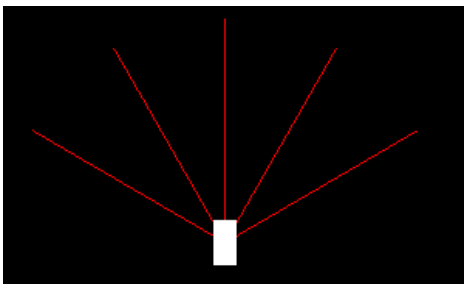

Figure 6: Car with FOV 120º.

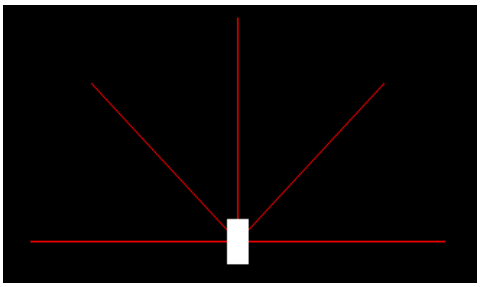

Figure 7: Car with FOV 180º.

4.3 Neural Network

The AI controls the car through a Feed Forwarding Neural Network, which does not contain cycles. This network is of fixed size and it contains 4 layers. The first one, the input layer, has a number of nodes equal to the number of rays the car has, in this research 5. Then, we have 2 hidden layers, where the first one has 4 nodes and the second one 3 nodes (the number of hidden layers and nodes has been decided to be like this after some testing). Finally, the last one, the output layer has 2 nodes, and they are used to know to which side the car has to turn. Apart from that, every node of each layer is connected to every node of the next layer, and also the values of the biases of the nodes and the weights of the links are normalized inside the range of [-1,1].

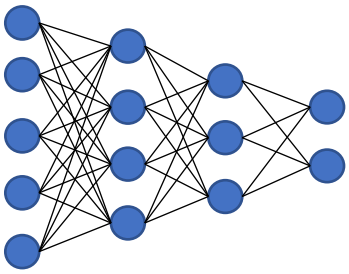

Figure 8: Neural network used in the research.

For the hidden layers, the activation function the nodes use is the following one:

$$f(x) = \begin{cases} 0, & x < \text{Bias} \\ 1, & x \geq \text{Bias} \end{cases} \quad (1)$$

Where x is the value the respective node gets fed with.

In this approach, the input layer gets fed with the collision times of the car rays. After all the computation of the hidden layer, the car decides to turn left or right with a fixed strength and then moves forward.

Listing 1 Function to turn the car using the AI

```
void MoveCar(Car * car, float dt)
{
    std::vector<float> ray_hit_times = car->GetHitTimes();
    std::vector<float> results = car->Feed(ray_hit_times);
    if (results [0] > results [1])
    {
         car->orientation.z += steer_strength * dt;
    }
    else
    {
         car->orientation.z -= steer_strength * dt;
    }
    //After this make car advance towards orientation.z
}
```

4.4 Genetic Algorithm

For the genetic algorithm side, we will try to simulate evolution for the Neural Network of the car, so in each generation we will take the best performers and use them for the following generation, for that, the following process happens:

1- Initial generation of cars NN with random weights and Biases
2- Make them race
3- When all of them crash, get the best performers
4- Create a new NN from the selected parents
5- Mutate the NN to create different NNs
6- Create a new generation with those NNs

The process stops when we consider it necessary. Also, we can record the Neural Network of the best performer or the one of the last winner if we want to start another session using that Neural Network.

4.4.1 Mutation

As the used Neural Network is of fixed size, we are not going to add/erase nodes of the Network, and neither we are going to connect/disconnect nodes, so the only things that change (the genes) are the biases of the nodes and the weights of the links.

In this research, two types of mutations are done, the soft ones and the hard ones. The soft ones change a value slightly to avoid getting the output affected a lot. The hard ones generate a new random value between [-1,1] and they replace the respective weight/bias by that value.

The probability for a mutation to happen is user dependent, but the probability for a soft or a hard mutation to happen is computed dynamically.

$$\text{Soft} = \frac{\text{crossed interest points}}{\text{total interest points}} \qquad \text{Soft} \in [0,1] \qquad (2)$$

$$\text{Hard} = 1\text{- Soft} \qquad \text{Hard} \in [0,1] \qquad (3)$$

4.4.2 Parent Selection – Crosspoint

After all the cars of a generation have crashed, new cars have to be selected to act as parents for the next generation. For this purpose, the cars are ordered from biggest to lowest heuristic value, and this heuristic value is computed taking into account the number of objectives crossed (in the correct order) and the inverse of the distance to the next objective midpoint.

$$\text{Heuristic value} = \text{crossed objectives} + \frac{1}{||\text{position - next midpoint}||} \qquad (4)$$

Once we have the parents ordered, we select a specific amount of them (specified by the user) to compute a crossed Neural Network. As our genes are the biases and the weights, these values will be used to compute the crossover points where we will change from one parent to another when crossing the neural networks. The crossing process will be to first set the links of one layer to the next one, then the biases of the next layer, and then follow again with the next links.

$$\text{Crossover points} = \frac{(\text{number of nodes} + \text{number of links})}{\text{number of parents}} \qquad (5)$$

5 Tests - Observations

For this research, some tests were made taking into account the different parameters that have been mentioned in the paper. For the following examples, the program was executed 5 times (for each table row) with some fixed parameters and with other ones changing. The fixed parameters were: a mutation probability of 7% and 20 cars per generation.

Table 1 Training in first map [Figure 1].

Used FOV	Number of parents	Average generations to learn
60º	1	14.2
60º	2	36
120º	1	3.8
120º	2	8
180º	1	6.8
180º	2	11.6

Table 2 Training in second map [Figure 2].

Used FOV	Number of parents	Average generations to learn
60º	1	258.2
60º	2	80.2
120º	1	37.6
120º	2	47.2
180º	1	28.6
180º	2	15.2

Table 3 Training in third map [Figure 3].

Used FOV	Number of parents	Average generations to learn
60º	1	153.8
60º	2	225.4
120º	1	100.8
120º	2	80.2
180º	1	128.2
180º	2	99.4

After looking at the examples, we can get some interesting information. First of all, regarding the FOV, we can see how the sweet spot is between 120º and 180º and how when we go lower than 120º the learning is way slower. Secondly, we can see how when the circuit gets more complex it is more beneficial to have a crossed Neural Network of 2 parents instead of choosing the Neural Network of the best car. Another thing that is obvious but still makes sense is that when the circuit is harder the AI takes more time to learn.

Other worth mentioning things that were of interest during the development of the project are the possibility of using a trained Neural Network on one map onto another map and how the car evolution depends on randomness.

Regarding the first thing, one curiosity observed was that training a car in a simple circuit would not work to make it race in a harder circuit, but training it in a complex one will “usually” make it work in a simpler one. The problem here is that with this method,

the car learns to react to specific situations/patterns to not crash, but it does not learn that crashing is a bad thing at all, because the car with better performance is selected and mutated instead of making it learn directly. So what usually happens is that whenever it encounters a pattern that has not been found before it will (probably) crash, so this method is not very good for the generalization of learning how to drive, instead is good for making it able to learn how to drive in a specific environment.

Second of all, as Neural Networks are complex graphs and mutations can change the values of the biases and weights, a bad initial generation could make the learning much slower and a bad mutation could make a very advanced car mesh up completely and go back in the evolution. This is not something that could be handled easily with this method, one solution that was thought of was to reduce the probability of mutation whenever the car completes a lap, but still, it is not the perfect solution.

6 Conclusion

As we saw through this paper, using Neural Networks and Genetic Algorithms is good for making a car learn how to drive in the circuits that is trained on, but not to make it drive in all of them. It is a process that takes a long time of testing and if you do not have enough luck, it could lead to catastrophic results.

However, the power and possibilities of these methods could be extended to a lot of other modern problems that do not have a conventional solution. I think that the usage of this method in other areas could lead to very interesting results and I encourage the readers to try these mechanisms for solving more problems and for the shake of research itself.

7 References

[Code Bullet 19] Code Bullet. 2019. A.I. Learns to Drive. Youtube video.

https://www.youtube.com/watch?v=r428O_CMcpI&t=871s