Self driving car

Tóth Noémi, Farkas Balázs

April 2020

1 Abstract

In this project we are going to teach two cars how to drive themselves.To achieve this, we will use neural networks, and genetic algorithm to train the cars in a 3D environment. [1] Our goal is to compare two different neural networks in order to check which one performs better.

2 Introduction

We decided to use Unity game engine, due to its simple editor and great physic mechanics. For implementing the networks, the genetic algorithm and the car controls, we used C# scripts, since it is highly supported and recommended in Unity. We are going to test 2 different cars, in the following, let's call them Green and Red. While training the cars, it is really important to train them on an appropriate track. It means, that the track should contain straight parts, left and right turns, or the cars will not learn how to deal with every situation.

3 Methods

3.1 Neural Networks

Neural networks [2] are a set of algorithms, that return outputs according to different inputs. The neural networks are usually trained before being used, but we are going to do it differently, and train the network during run time with genetic algorithm. Every time a car crashes, a new random neural network is generated. For Green's network, we gave 3 values as inputs. To get these values, we used sensors to check how far the car is from an object. So we placed 3 sensors on Green, one that checks what is directly in front of the car, and two additional sensors that are diagonals, rotated by 45 degrees to the left and right. For Red's input however, we used 5 inputs,we added two more in between the sensors. By doing this, we were expecting cleaner turns, and better movement.

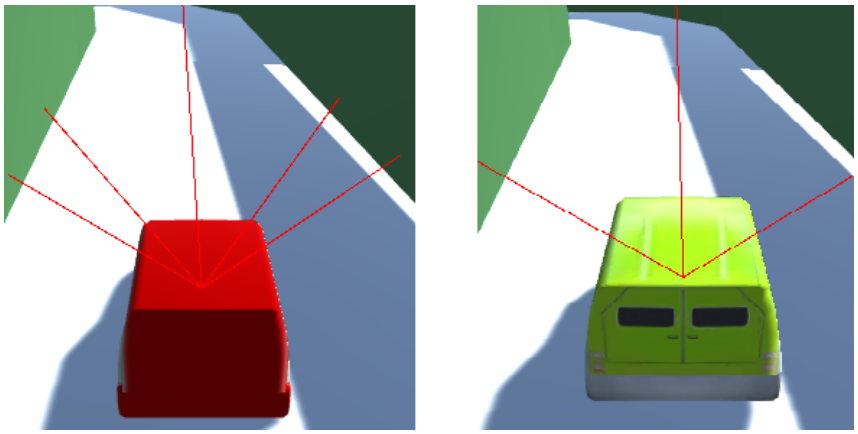

Figure 1: Red's and Green's sensors

Keep in mind, that at the beginning all neural networks are randomly generated. Due to this fact, it is nearly impossible to make any predictions about which car will do better, because there is a chance, that on the first try a perfect neural network will be generated, completing the track on the first try. On the other hand, in the first generation all neural networks could be completely useless, meaning that the cars will never, or only after a very long time learn how to complete the track.

For the hidden layers, we used one layer with ten neurons for both cars. We did not see a reason to build in more layers. The number of the neurons are not professionally calculated either, ten just seemed an appropriate amount, and turned out to be good.

The outputs are the same in both networks. The are two outputs, an acceleration, that tells the car how fast to go, and a turning value, that makes the car turn left or right. The acceleration value is between 0 and 1, so we used a Sigmoid function to Normalize the output data. The turning value however, is between -1 and 1. -1 means that the car has to go left, 1 means the car has to go right. To achieve this, we used the Tanh function.

$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$
$$Tanh(x) = \frac{e^{x}-e^{x-1}}{e^{x}+e^{x-1}}$$

3.2 Genetic algorithm

A genetic algorithm [3] is inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators. We used this algorithm to train Green's and Red's neural network. The algorithm has three very important methods, these are the selection, cross-over and mutation. Each car has a fitness value, that determines how well has it performed. [4] Fitness is calculated by the distance the car has successfully travelled without crashing, multiplied by the average speed it has gone with. The multipliers are different, we prioritized the distance travelled rather than the speed, so the distance travelled multiplier is higher than the acceleration one. Otherwise, a car that crashed into a wall but went very fast, would be considered "better" than another car, that has completed even a turn, but very slowly. We had to keep in mind, that the neural networks are randomized, so there is a chance that a car's acceleration value will be set to zero, meaning that it is going to stand still, achieving 0 fitness, but avoiding crashes as well. So we implemented a timer, that counts how long had a car been idle. If it can not achieve 40 fitness within 20 seconds, the network will be considered as a failure, so we reset the car with a new random network. On the other hand, once a car shall be created to complete the track. But we do not want to stop here, we want even better cars. So if a car performs "too good", like it has completed the track a couple of times, we save the network to the genepool, and reset it. This car will be used as an example to the other cars, and will be mutated in the next generation, to get better and better cars.

3.2.1 Selection

When working with genetic algorithm, we make new generations according to the best performed species in the previous generation. Therefore, we need to get the best species. We made a genepool, that stores the neural networks that are assigned to be mutated. We select the best 8 neural networks from a generation, and put it into the genepool, but in an interesting way. The cars that performed better are saved more times, than the cares that still performed well, but only like the 8th well. By doing this, there will be a higher chance to get better cars during crossover.

3.2.2 Crossover

Crossover is the way we want the genetic algorithm to mutate our species. In our example, we are generating 85 cars in a generation, and we are choosing the eight best performed cars through the selection. According to these cars, we generate the next generation, but in an interesting way. We only mutate nearly the half of the initial population (85 in every generation) of the cars in a single generation from two randomly chosen saved car, the rest are randomized. That means, from the 85 cars in a generation, only 39 is generated from the previously chosen best performed cars. This might bring up some questions, such as: Why not generate the whole population according to the very best car? Why are we only generating the half of the population, and randomize the rest? Well, genetic algorithms are all about randomness. If we only generated neural networks from the #1 car, maybe we would never get them through the track. That's because a car might be good enough, but another car can be just as good, but in a different way. And this is the whole point of it, because if we combine these two cars, we can get an even better car. That would never be possible if we only generated cars according to the best performed one. And why not generating 85 species instead of 39? The reason is the same. Cars are ought to learn from each other, that is the beauty of the genetic algorithm.

3.2.3 Mutation

The mutation is the actual way we combine the neural networks, and get a new, hopefully better one. As the previous sentence says, we can not be sure that a mutated car will perform better, than a randomly generated one. The randomly generated neural networks also have a chance to get mutated, but a very low chance, 5%. The Reason for it being so small, is that we do not want every car to get mutated, we need random neural networks as well. The mutation works like in human biology: a car is generated from two randomly chosen best performing neural networks. There are 50-50% chance on every bias and weight to be assigned to the child. So the mutated cars in the new generation are created by their parent's values. Imagine it like you have your mother's eyes, but your father's hair. It is the same, the neural networks get their parents' values randomly.

4 Results

In order to compare the two neural networks, we need some kind o measurement. We wanted to know, which neural network goes through the track first without any mistakes. Since the algorithm is based on randomness, we got some interesting results. We made 25 test runs, in which 3881 red and 4491 green car were generated. In this table we summarized, which generation and which genome was the car in, when it successfully completed the track, and which one came out victorious. The later is marked with green. During measuring, the outcomes could be really diversified.

	RED			GREEN		
No	Generation	Genome	Total	Generation	Genome	Total
1.	2	60	230	4	11	351
2.	0	30	30	0	14	14
3.	0	9	9	0	29	29
4.	3	9	264	3	5	260
5.	0	62	62	0	82	82
6.	6	8	518	6	40	550
7.	1	6	91	0	68	68
8.	1	23	108	1	66	151
9.	1	30	115	1	39	124
10.	4	29	369	3	22	277
11.	1	20	105	2	26	196
12.	2	11	181	2	8	178
13.	1	19	104	1	20	105
14.	1	14	99	1	44	129
15.	2	14	184	2	23	193
16.	1	37	122	1	27	112
17.	1	29	114	1	25	110
18.	0	63	63	1	0	85
19.	1	33	118	1	69	154
20.	4	37	377	7	1	596
21.	1	10	95	1	9	94
22.	0	44	44	0	36	36
23.	2	11	181	3	41	296
24.	2	43	213	3	4	259
25.	1	0	85	0	42	42

Table 1: Results of the test runnings

We noticed, that sometimes the track was completed in a really short time, after one generation, or even in the 0th generation, but in some cases, the cars could only complete the track after 7-8 generations. In case of Red's results, the average completion time was in the first generation, 69th genome, while Green's average completion time is a bit higher, in the second generation, 9th genome. Al tough Green could sometimes finish a lap in a shorter period of time, it took it more genomes to learn, so more car were destroyed, in less time. So, Red takes fewer genomes to learn the track, it makes less mistakes, and completes the track slowly, but surely, while Green seems to like learning from it's ow mistakes, sacrificing itself a lot more time in order to learn. According to the table, out of 25 test runs, 13 times Red came out victorious, and Green won 12 times. It can be seen, that the two neural networks almost completed the track with the same efficiency.

5 Conclusion

In conclusion, both of the neural networks are capable to complete the track, are able to overcome any problem, recognise walls, and breach further avoiding any obstacle. The neural network with only three inputs is only a bit behind the on with five inputs. According to the Law of large numbers though, Red's win rate would be higher and higher, if we ran the tests like a million times. But that would take several month to measure, so we just made 25 test runs, and the results are visible.

References

[1] Suryansh S. Genetic algorithms + neural networks = best of both worlds, March 26 2018. Neural Networks coupled with Genetic Algorithms can really accelerate the learning process to solve a certain problem.

[2] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, USA:, 2015.

[3] Vijini Mallawaarachchi. Introduction to genetic algorithms, July 8 2017.

[4] Louis. Was darwin a great computer scientist?, August 29 2017. How evolution taught us the "genetic algorithm".