Application of Neuroevolution in Autonomous Cars

G. Sainath, S. Vignesh, S. Siddarth, and G. Suganya

Abstract With the onset of electric vehicles, and them becoming more and more popular, autonomous cars are the future in the travel/driving experience. The barrier to reaching level 5 autonomy is the difficulty in the collection of data that incorporates good driving habits and the lack thereof. The problem with current implementations of self-driving cars is the need for massively large datasets and the need to evaluate the driving in the dataset. We propose a system that requires no data for its training. An evolutionary model would have the capability to optimize itself towards the fitness function. We have implemented neuroevolution, a form of genetic algorithm, to train/evolve self-driving cars in a simulated virtual environment with the help of Unreal Engine 4, which utilizes Nvidia's PhysX Physics Engine to portray real-world vehicle dynamics accurately. We were able to observe the serendipitous nature of evolution and have exploited it to reach our optimal solution. We also demonstrate the ease in generalizing attributes brought about by genetic algorithms and how they may be used as a boilerplate upon which other machine learning techniques may be used to improve the overall driving experience.

Keywords Neuroevolution · Neural networks · Genetic algorithm · Generation · Fitness · Selection · Crossover · Mutation

1 Introduction

The society of automobile engineers (SAE) has coined six different levels of autonomy beginning at level 0, the absence of any autonomy, to level 5, complete autonomy requiring no human intervention whatsoever. Currently, many luxury vehicles possess level 3 autonomy in terms of cruise control and active lane control, and a handful of vehicles possess level 4 autonomy. Level 5 autonomy in cars is still under research and development. The main barrier to attain this level of autonomy is the task of collecting data and the lack thereof. Although a deep model can be extremely

G. Sainath · S. Vignesh · S. Siddarth · G. Suganya (✉)
Vellore Institute of Technology, Chennai, India
e-mail: suganya.g@vit.ac.in

R. J. Kannan et al. (eds.), *International Virtual Conference on Industry 4.0*,
Lecture Notes in Electrical Engineering 355,
https://doi.org/10.1007/978-981-16-1244-2_26

adept at learning and generalizing features, it can only learn what it sees. Humans can learn from different scenarios. Essentially, even if it learns to navigate through a busy street, it may not be able to correct oversteer or understeer due to several factors such as poor roads and tyre wear causing a loss of traction, which may not have been accounted for in the training dataset. That is why an evolutionary approach would solve these issues. What if the car could learn to drive on its own, via trial and error, over countless generations? It would have trained, evolved to overcome such edge cases and scenarios, and would know exactly what to do once it detects wheel spin or any form of loss of traction/grip.

2 Overview of Existing Systems

2.1 Neural Networks (Supervised Learning)

A neural network is an interconnected network of neurons, also called nodes. Each neuron has a set of output edges that activate based on the resultant value obtained from the weighted inputs it received from the previous layer (Fig. 1).

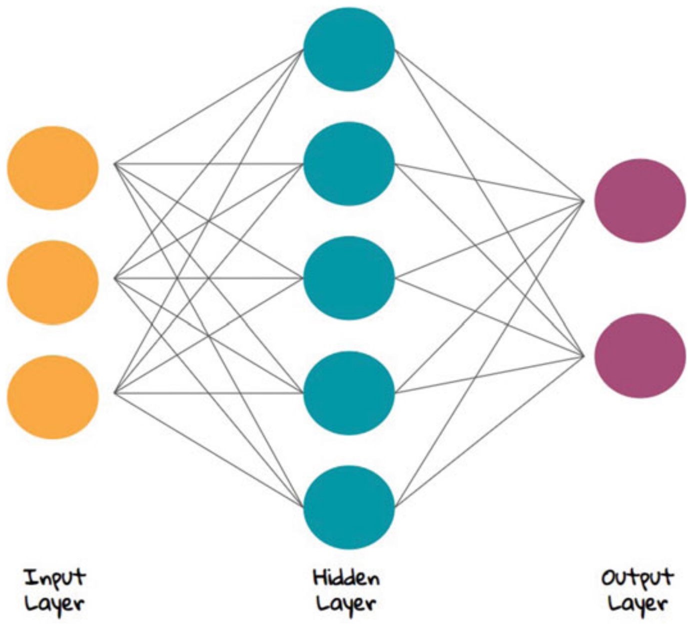

Fig. 1 Topology of an artificial neural network

In a supervised learning approach, we would have a list of attributes or features as our inputs and a list of targets as our outputs. We would then have to use back-propagation to train our neural network to correct its weight to suit our target and increase its accuracy.

So, in a situation where it is difficult to obtain a dataset large enough to train the neural network to a certain degree of accuracy, we will face problems arriving at our optimal solution. This is especially true in the scenario of self-driving cars, where large corporations like Nvidia use 1000 h of driving data to train their vehicle to navigate the roads. In such scenarios, we could adopt an evolutionary technique that requires no datasets and train our model in a simulated environment.

Although deep neural networks utilizing convolutional layers have performed extraordinarily in several scenarios, the problem arises after the fact that it can only learn what it is shown or taught. Since there are countless more possibilities of things that can happen on the road, which cannot be accounted for in the driving data we gather. Moreover, such networks are notoriously prone to over-fitting.

2.2 Reinforcement Learning

Reinforcement learning is another critical area of research in autonomous vehicles, where an agent learns to accomplish a task by gathering experience by itself, rather than through a supervised dataset. The basic gist of the algorithm is that an agent granted a reward when it performs an action that is desirable in the current scenario and gets punished if it does something undesirable. Although this form of carrot and stick approach seems to be how we, as individuals learn, the key drawback of this algorithm is that the agent has no prior experience whatsoever. We humans learn pretty quickly through this approach due to the generalization of a multitude of experiences that we have gathered from birth till date. This is not the case for the agent, and so it takes quite a while, depending on the complexity of the problem, for the agent to gather enough experiences in order for it to determine whether a certain action is desirable or not [1, 2] (Fig. 2).

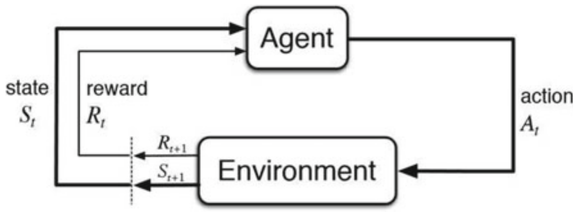

Fig. 2 Basic flow of reinforcement learning

3 Proposed System

Neuroevolution is a genetic algorithm that is used to evolve artificial neural networks. In this model, each species of a generation has a brain (the neural network) that has a set of genes (weights). In the beginning, all species of the population have random weights and hence perform random actions. It is through serendipitous discovery that a certain species gets closer to our solution. We select this species based on a fitness function and pick similarly performing species to perform crossover. After crossover, we mutate this gene and pass it on to the next generation. Owing to the nature of this sort of evolution, genetic algorithms are easily parallelizable as the actors (neural networks and their respective vehicle) of the population are individual entities independent of each other [3].

So the entire genetic algorithm can be summarized to three key processes:

- Selection: We select the best species of the generation based on the fitness function.
- Crossover: We crossover the genes of the population to converge onto our solution.
- Mutation: We mutate the genes, in the hope of a better solution, of the selected species following crossover (Fig. 3).

We can see that mutation and crossover seem a little opposite to one another. Mutation randomizes the weights of a certain percentage of neurons while crossover tries to converge them. There is a trade-off here between exploration and exploitation. Exploration via mutation is exploring new gene sets out of a hope that something new can lead to promising results whereas exploitation via crossover is taking what you

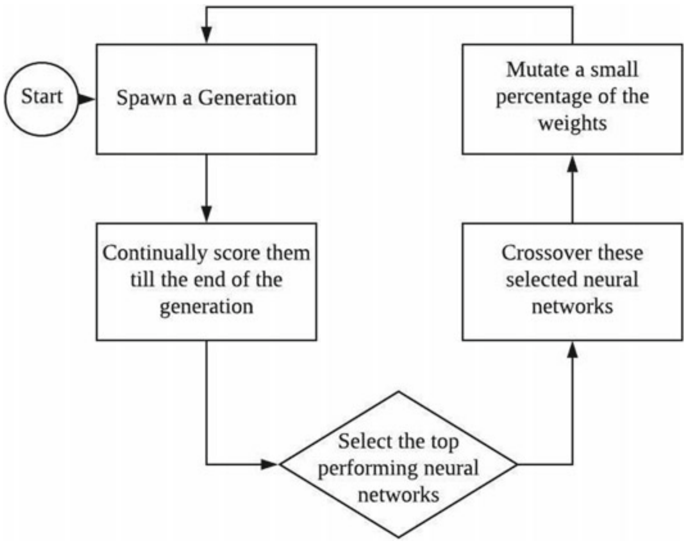
Start
Spawn a Generation
Continually score them till the end of the generation
Select the top performing neural networks
Crossover these selected neural networks
Mutate a small percentage of the weights

Fig. 3 Simple pipeline of neuroevolution

learned and using that information, combining the best, to inform newer decision-making processes.

4 Implementation

We chose to simulate neuroevolution using Unreal Engine 4, which is a game engine that utilizes Nvidia's PhysX Physics Engine to replicate real-world like vehicle dynamics, which is essential if we plan to transfer the learning that has happened in this environment.

Compared to using simulators such as CARLA, which was also built on the same engine, we have a lot more freedom when we build the whole environment from the ground up, in terms of level design, vehicle physics, frame times (time dilations) and overall gives more power to the user.

4.1 Vehicle Dynamics

FWD Layout—Since most vehicles these days in the low to mid-tier range are front-engine, front-wheel drives (FF), we chose this as our vehicle layout, and for the differential, we went with a limited-slip differential (LSD) that prevents wheel spin, which is getting more and more common these days. The transmission of the vehicle is set to automatic. The suspension settings have also been altered so that it favours understeer rather than it oversteer, as most manufacturers do these days, as it is easier to correct understeer. Weight transfer and tyre traction are also essential aspects that dictate the vehicle's physical handling and are simulated accurately.

RWD Layout—We also wanted to observe how this approach would fare on a more difficult layout which is harder to control, which is the front-engine, rear wheel drive (FR), also the typical sports car layout, as they more prone to over-steering and sliding through corners without proper throttle control and adequate countersteering. The suspension of this layout has also been altered so that it favours oversteering behaviour rather than understeer.

4.2 Neural Net

Each vehicle that we simulate has a brain that controls the values for the throttle pedal, the brake pedal, and the steering angles directly. This brain is our deep neural network which outputs a value from -1 to 1 for all the above inputs of the vehicle. The inputs to the neural network are the distances (normalized to 0–1) we obtained by tracing a point cloud around the vehicle. We also feed in the current speed of the vehicle (normalized to its maximum speed) and the angle between the velocity of the vehicle and its forward vector, which provides the neural net information about in which direction the car is sliding towards, if or when it does.

4.3 Genetic Algorithm

The genetic algorithm in this simulation is a higher-level entity that oversees the processes responsible for selection, crossover and mutation. It controls the mutation and crossover rates and is responsible for spawning and tracking all the features of the entire vehicle species for each generation of the population. Initially, in the first generation, all the weights of all the neural networks are initialized to random values, and it is through serendipitous discovery aided by the fitness function that we converge on to a solution through selection and crossover and also search for a better solution through mutation.

4.4 Working

In the first generations, all the weights of all the neural networks in the vehicles are initialized randomly, so they have no clue what to do when they are spawned and hence move randomly. To remove poorly performing agents, which is a crucial part of Darwinian evolution, "survival of the fittest", we de-spawn vehicles that crash into obstacles and guard rails or those that do not reach a certain threshold score within a predetermined period of time.

Pipeline:

1. Genetic algorithm entity spawns a vehicle population
2. Select top-performing vehicles based on the fitness function
3. Perform crossover by the weighted average of their weights with respect to their relative fitness in terms of the population
4. Mutate a small percentage of the weights by setting them to random values
5. Spawn the next generation of vehicles
6. Repeat for n generations till satisfactory behaviour is observed (Fig. 4).

Selection For each vehicle, the distance travelled (in the direction of the course) each frame, which we call the score of the neural net, is calculated as:

$$\Delta d = v \times \Delta t \tag{1}$$

Δd = distance travelled that frame
v = instantaneous speed
Δt = frame time.

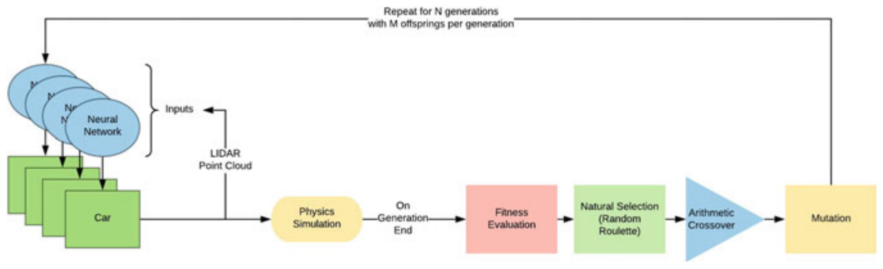

Fig. 4 Neuroevolution architecture

In order to prevent over-correcting behaviour and that it does not game the fitness score, we increment its score only when the angle between the velocity vector and the car's forward vector is less than a threshold value which we set as 10°.

From this, we calculate the net score each frame, which is the total distance travelled (until it de-spawns) and is calculated as:

$$\text{score} = \sum \Delta d \tag{2}$$

At the end of each generation, the relative fitness of each neural network is calculated as:

$$\text{fitness}_i = \frac{\text{score}_i}{\sum_{j=1}^{p} \text{score}_i} \tag{3}$$

where

fitness_i = relative fitness of the current neural network
score_i = total distance travelled by the vehicle
p = total population of the generation.

Now, for spawning the next generation of vehicles, we pick the top n vehicles with the greatest fitness (n could be selected arbitrarily, we chose it to be 1/10th of the population, p).

Crossover We now perform an arithmetic crossover of these n species by weighted addition of their weights with respect to their fitness. For each connection in the neural network, the weight of the connection after crossover is calculated as:

$$\text{new } w = \sum_{i=1}^{n} (w_i \times \text{fitnees}_i) \tag{4}$$

where

fitness_i = relative fitness of the current neural network
w_i = weight of a certain connection in the neural net

n = the top selected species of the generation.

Mutation Once we perform crossover for about 80% of the weights in the neural network, we then move on to mutation which is typically done to about 20% of the weights by using a random function on the weights.

5 Results

Within the simulated environment, we have observed that the population over several generations has evolved to not crash into obstacles, and also through sheer randomness have decided to stick to one side of a lane in certain simulations. They also perform advanced traffic management techniques such as zipper merges. A zipper merge is when a car continues to stay on the lane even after a blockade is located. They merge into the free lane only once they are close to the blockade in order to prevent traffic congestion that would occur if everyone stopped using that lane entirely (Figs. 5, 6, and 7; Table 1).

Modeling using various simulations, iterating and variating different parameters, are that when the fitness function is simple, it generalizes the course pretty quickly and is able to navigate it well over very few generations. But, when we alter it so that it favours a certain style of cornering or maintaining a certain amount of speed, it takes drastically more generations for it achieve this sort of specialization. This interpretation is backed by the discrepancies seen in the number of generations it took for the front-engine, rear-wheel drive (FR) layout compared to the front-engine, front-wheel drive (FF) layout (Fig. 8).

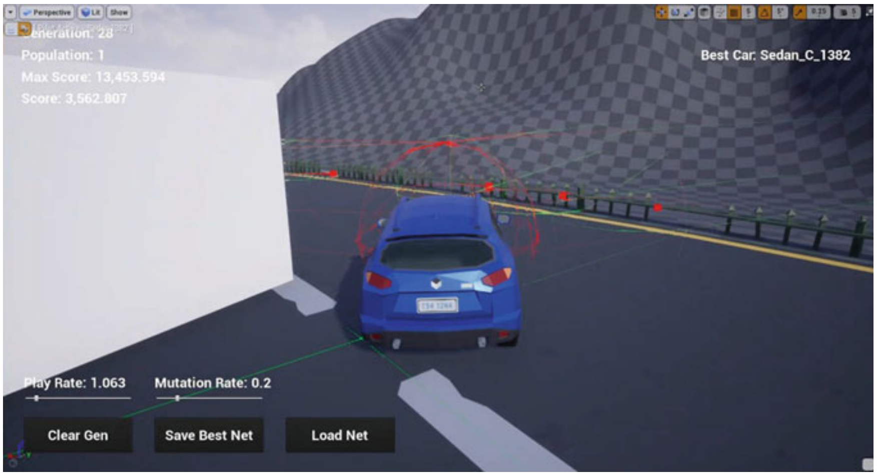
Generation: 28
Population: 1
Max Score: 13,453.594
Score: 3,562.807
Best Car: Sedan_C_1382
Play Rate: 1.063
Mutation Rate: 0.2

Fig. 5 Neural net switches lane only at the very verge of colliding on to the obstacle (white wall to left)

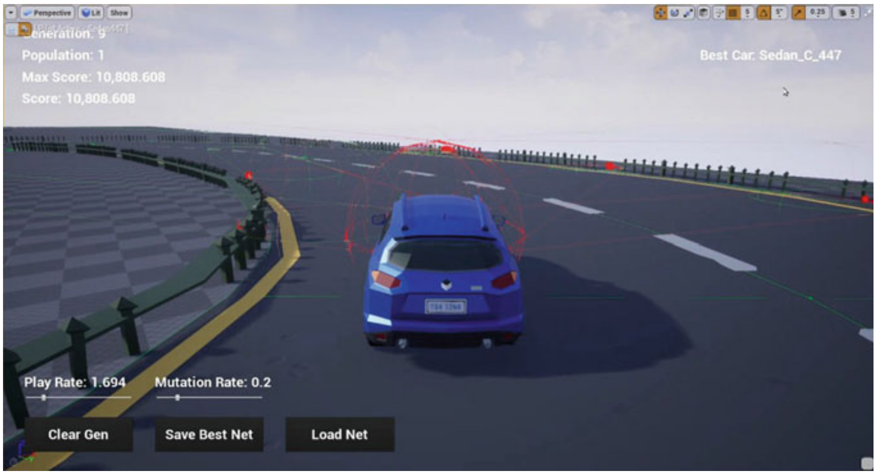

Fig. 6 Neural net has decided to stick to the left lane on the road through sheer randomness which can be nurtured by altering the fitness function

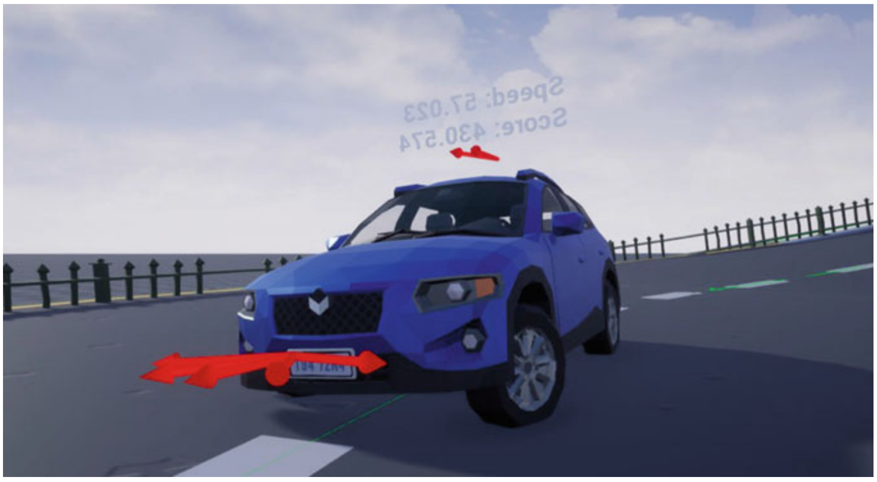

Fig. 7 Neural net has learned how to countersteer and control the car on the onset of oversteer

6 Conclusion and Future Work

Based on the results we have observed above, we can come to the conclusion that genetic algorithms such as neuroevolution can speed up the initial phase of generalizing features several fold compared to traditional techniques such as back-propagation which place the prerequisite of procuring a massive dataset for training so as to not overfit the solution. But once the network attains the basic cognitive

Table 1 Generations taken by the neural nets to evolve enough to navigate the entire course without crashing

Layout	Crossover rate (%)	Mutation rate (%)	Generation	Population
FR	80	20	97	4850
FR	80	10	225	11,250
FR	90	20	145	7250
FR	90	10	171	8550
FF	80	20	24	1200
FF	80	10	26	1300
FF	90	20	38	1900
FF	90	10	12	600

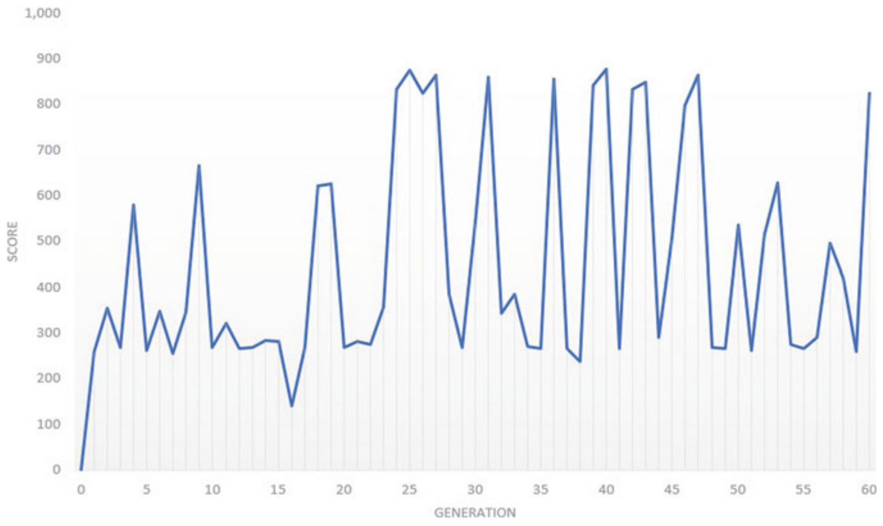

Fig. 8 Once one the neural nets hit a peak, it is able to constantly replicate the peaks which is an indication of evolution

abilities for driving, it can be further improved upon through reinforcement learning techniques such as deep Q-learning [4, 5], since it has already gathered a plethora of experiences over several generations, which is one of the key barriers slowing down reinforcement, since now we can quickly jump to the phase where the focus is more on obtaining as many rewards as possible rather than the initial phase of gathering experience where the agent primarily tries to just not get punished for its actions.

References

1. Saez Y, Perez D, Sanjuan O, Isasi P (2008) Driving cars by means of genetic algorithms. In: Rudolph G, Jansen T, Beume N, Lucas S, Poloni C (eds) Parallel problem solving from nature – PPSN X. PPSN 2008. Lecture Notes in Computer Science, vol 5199. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-87700-4_109
2. Such FP, Madhavan V, Conti E, Lehman J, Stanley KO, Clune J (2017) Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. ArXiv, abs/1712.06567
3. Stanley KO, Clune J, Lehman J et al (2019) Designing neural networks through neuroevolution. Nat Mach Intell 1:24–35. https://doi.org/10.1038/s42256-018-0006-z
4. Stanley KO, Miikkulainen R (2002) Efficient reinforcement learning through evolving neural network topologies. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation (GECCO'02). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 569–577
5. Iglesias Rodriguez R, Rodríguez M, Regueiro C, Correa J, Barro S (2006) ICINCO 2006, Proceedings of the Third International Conference on Informatics in Control, Automation and Robotics, Robotics and Automation, Setúbal, Portugal, August 1-5, 2006, pp 188–195