Training Genetic Neural Networks Algorithms for Autonomous Cars with the LAOP Platform

Jihene REZGUI‡, Léonard OEST O'LEARY‡, Clément BISAILLON‡, Lamia CHAARI FOURATI†

‡ Laboratoire Recherche Informatique Maisonneuve (LRIMa), Montreal, Canada

† Laboratory of Technology and Smart Systems (LT2S), University of Sfax, Tunisia

Digital Research Center of Sfax (CRNS)

jrezgui@cmaisonneuve.qc.ca lamiachaari1@gmail.com leonard.oest.oleary@gmail.com

Abstract- **The challenge with self-driving cars is to create a model that converts sensors data (such as cameras or proximity sensors) into actions. This way the car can react to its changing environment and make the right *decisions*. In the literature, Neural Networks is the most promising technique used to parse these sensors data. A well trained and designed neural network can take the sensors values and output the right actions. In this paper, we introduce a Way to train Efficiently Neural Network with Genetic principle, called WENNG. Moreover, we propose a comparative study between all the variations of WENNG to highlight the best-performing ones. To evaluate our WENNG training variation, we implement two well known neural network algorithms: the FullyConnected one and the NEAT algorithm. Through extensive simulations, we demonstrate that the Natural Selection WENNG outperforms the Greedy WENNG at training the genetic neural networks with a low mutation rate. Finally, we show that an IMproved version of NEAT called IMNEAT, minimizes twice the number of generations to reach the maximum fitness value compared to the traditional NEAT algorithm.**

Keywords: Neural networks, evolution, autonomous cars, natural selection, FullyConnected, learning algorithm;

I. Introduction

Nowadays, autonomous vehicles (AV) gain a great focus from civilians, automotive constructors and transportation stakeholders. Accordingly, AV, Internet of vehicles, and the connected smart cars will be the main actors of smart transportation system within smart cities. AV could be better at preventing accidents than humans, since they can react faster to disturbance and they can do precise evasive maneuvers. AVs make more reliable decisions which lead to faults avoidance that could cause accidents. Besides that, AVs will get better traffic flow regulation than humans because AV ride using proper traffic rules, making smooth and congestion free traffic. The AV basic model involves front-facing cameras, rear cameras, radar, digitally controlled sub-systems, long-range ultrasonic sensors located around the car and many sensors and actuators embedded within the vehicle as well as intra-vehicles and inter-vehicles networks [2]. All these sensors will gather in real-time the required data concerning the vehicles environment which are fused into a learning network predicting the vehicle's response. However, autonomous vehicle faces diverse challenges before its standardisation. In this context, Intelligence artificial is the key paradigm that will enable the researchers, engineers and developers to build safe AV. Therefore, a huge effort is required to find the better ways to make the machine learn to drive safely and efficiently. Now, the most reliable way to achieve artificial intelligence in computing is to use various known algorithms as artificial neural networks. Multiple challenges arise with those kind of algorithms. For example, in order to train the neural networks we need to simulate cars in an environment that is as close as possible to the real world in order to be able to transfer the algorithm to a real car when it is done learning. Another problem with this approach is that there are multiple variations of the artificial neural network algorithm each tested using different techniques. In our previous work, we proposed a platform called LAOP [3] to easily compare different neural network algorithms in the context of autonomous cars. Using our platform, scientists can collect comparative data between multiple neural network algorithms to drive cars. Note here that we are differencing the Artificial Neural Network (ANN) from the learning algorithm. The learning algorithm is the one that will perform the optimisation on the ANN (for example, the backpropagation algorithm is a learning algorithm). Using the LAOP platform, we compare different variation of a genetic learning algorithm that we called WENNG on two well known algorithms : NEAT [5] and FUlly COnnected Neural Network (FUCONN).

Our contributions can be summarized as follows: (1) We present the global working of our Way to train Efficiently Neural Network with Genetic principle named WENNG and its implementation in the LAOP platform [4]; (2) We introduce the two variations of WENNG :Natural Selection named NS WENNG and GReeDy WENNG named GRD WENNG; (3) We implement two neural networks algorithms : NEAT and a FUCONN to evaluate the performance of WENNG and its variation;(4) We discuss our comparative study using all the WENNG variations to train NEAT and FUCONN respectively and (5) We demonstrate that the Natural

Selection WENNG outperforms the GRD WENNG with a low mutation rate while the IMNEAT algorithm outperforms the known NEAT algorithm.

Section II provides a brief overview of the related work and compares them to our proposed scheme WENNG. In section III, we briefly explain how our LAOP[4] platform works. In section IV, we present how we implemented some well known algorithms to test WENNG on it. Section V shows the simulations results. Finally, conclusions are drawn in Section VI.

II. Related work

Several schemes training neural networks have been proposed to handle self-driving cars challenge. To the best to our knowledge, these can be broadly classified in two categories: (a) supervised learning [6,7,10] and (b) reinforcement learning [1,5,8-9,11].

a. Neural networks trained with supervised learning

Dean A. Pomerleau [7] was one of the first to use supervised learning in autonomous driving. He proposed to train a fully connected neural network to predict steering wheels angles depending on road images. As he did not have access to a dataset containing steering wheels angles, he trained his neural network in a simulated environment. The authors in [6,10] used a Convolutional Neural Network (CNN) to predict steering wheels angles from raw images with great success. They used a dataset from images and steering wheels to train their neural network.

b. Neural Networks trained with reinforcement learning

Reinforcement learning has made their proof in solving multiple optimization problems, notably in games. This technique was used by DeepMind [8] to beat the best go player in the world. It was also used by OpenAI [9] to learn a bot to play dota 2.

There is a lot of techniques used to train neural networks such as Q-Learning [11] and its variants, but in this paper we concentrated ourselves on genetic reinforcement learning. In NEAT[5], the authors showed that a genetic algorithm can solve the pole-balancing problem with better results then the ones availables in that time. Recently, their research study, in [12], showed that genetic algorithms can be very performant even in hard optimization tasks.

c. Using a *genetic* reinforcement learning: What is the advantage compared to both categories?

To train the algorithms, in our paper, we use a genetic reinforcement technique that we called WENNG. The advantage of a reinforcement learning approaches [1,5,8-9,11] is that less data is required to train the network. Supervised technique [6-7, 10] needs to have a set of good outputs for each time the network is used. In our case, this means that for each step of the simulation, we would need to tell the network what it should be the output. The problem is that there can be multiple manoeuvers to get to the finishing line and forcing a predefined path may not be the best way to make it learns. We also want to explore what are the capabilities of a neural network without explicitly telling him what is the right path to take. A reinforcement learning approach only needs reward function that tells how the neural network is performing. With this information, it can favorise the best-performing ones and eventually find a good neural network.

III. An overview of the improved LAOP platform enforced by new algorithms

This section will briefly describe LAOP. We introduced a preliminary version of this platform in details in our previous work [3]**.** The source code of the new implementation of LAOP platform is available in [4].

LAOP [3,4] is a platform made to compare and develop better artificial intelligence algorithms in the context of autonomous vehicles. It can simulate and train a neural network algorithm. It can then provide comparison data between the algorithms tested. This section is divided as following : we first describe how we implemented the car into the platform and then we describe how we divided the process of generations.

A. The car

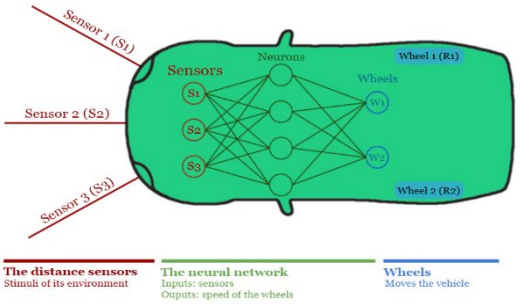

Figure.1. A car with its different parts. The sensors give information to the neural network that can dictates the values of the wheels.

The cars navigate through a simulated environment. This environnement is composed of two parts : (1) walls and (2) a starting position. The starting position is the point where the cars will spawn at the start of the generation. The walls are lines that the car must avoid. If the car comes in contact with one of these, its state is changed to *eliminated*, and it can no longer move.

A car is composed of three key parts : the sensors, the neural network and the wheels as shown in Figure.1. The sensors track information about the environment and transfer that data to the neural network. At the moment, only proximity sensors are implemented. They can tell the distance between the car and the walls surrounding it in a straight line. After receiving the data from the sensors, the neural network will do a calculation depending on its implementation and will assign a value to each of the two back wheels. The wheels then go at a

certain speed depending on the values returned from the neural network.

. **B. The simulation**

At the launch of the platform, the user can configure the simulation and select which algorithms that are going to be simulated. The user can also set specific settings to each algorithm. Therefore, the user can compare multiple variation of the same algorithm to gain valuable insight. As shown in [4], our LAOP platform can be useful for example to know the effect that the car density has on the learning process of an algorithm.

The simulation process can be described as follow: for each algorithm, a simulation batch is created; the simulation batch will simulate the same algorithm multiple time to reduce the error related to chance ; the simulation will play one generation after the other in order to make the cars learn. A generation contains a set of cars and it's the process of simulating all the cars. At the beginning of the generation, the cars are spawned at the starting location. For each car, the value of the sensors are fed in the neural network to get the value that each wheel should go. The new position of the car is then computed. This iteration continues until one of those conditions are met : (1) all the cars are eliminated, meaning they all hit a wall or (2) the time limit specified in the simulation settings is reached (this parameter is set to 60 seconds in our scenarios). When the generation ends, the cars go through the training algorithm WENNG to hopefully optimize the neural networks of the next generation.

III. WENNG, a Way to train Efficiently Neural Networks using Genetics

Our WENNG scheme uses three phases to make the set of car learn: (1) the evaluation, (2) the selection and (3) the repopulation as shown in Figure.2. The evaluation is done at the end of each generation, and it is the process of assigning a value to each car depending on their performance during the simulation. The selection is the process of eliminating the worst car depending on their fitness value. The repopulation is the process of repopulating the set of cars. After these three phases, a new generation is created with this new set of cars.

In the following part, we present two variations of WENNG : a Natural Selection Way of training Efficiently Neural Network using Genetics (NS WENNG) and a greedy WENNG (GRD WENNG). We will first present the way the cars are *evaluated*. This process is the same for the two algorithms. Then, we will present each of the algorithms independently.

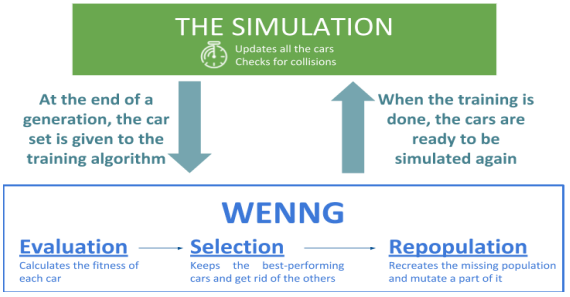

Figure. 2. The process of simulating consist of three phases: the evaluation, the selection and the repopulation.

A. The fitness function

During the simulation, we collect information about the cars. With this information a value is assigned to each car, called the fitness, that determines how good the car performed during the generation. Our implementation of the function that computes the fitness is the one displayed in (1).

$$f = d + x \qquad (1)$$

Where f is the fitness of the car, d is the *maximum distance from the start the car went* in pixels and x is the *total distance traveled by the car*, also in pixels. The reason why we chose this function over f=x is explained in [3].

B. Natural selection WENNG

After the attribution of the fitnesses, NS WENNG uses a weighted random distribution algorithm to eliminate the worst performing cars. This distribution favorises the cars with the best fitness by giving them a better weight then the others. Depending on this weight, half the population is eliminated. With this method, even the best-performing cars still have a chance to be eliminated. This respect the principles of natural selection. Even a well adapt specie still has a chance to die. More details about the implementation of the weighted sum is available in [3] .

After the elimination process, the algorithm must repopulate the missing cars for the next generation. Each neural network algorithm (like NEAT or FUCONN) must redefine a function called crossOver(). This function takes as inputs two neural network and returns one. The returned neural network must have similar properties then the two passed in parameters. NS WENNG then takes two random parents from the surviving set of cars and parse them into the function to create a new neural network. A new car with this newly created neural network will be put in the set. This will be done for each of the missing car in the set. Then mutations are applied randomly on the cars's network.

C. Greedy WENNG (GRD WENNG)

The greedy WENNG algorithm uses a much simpler approach to select the cars. It kills 80% of the worst

performing cars. Then, the algorithm puts the 10% best performing neural network in an array that will be kept for all the simulations. This array is used in the repopulation phase.

To repopulate the set, GRD WENNG first creates a sub-array containing the 20 best cars of this array. Then, for each missing car, it picks 2 random cars in the sub-array and reproduce them with the crossOver() function to create new cars. All the newly generate cars are forced to be mutated. The 20% that stayed in the array isn't mutated.

IV. Proposed algorithms

We implemented a few learning algorithms in the platform to compare them with each other. In this section, we will describe in better details how our algorithms are implemented. We propose two learning algorithms : a fully connected neural network with some genetic and the NEAT algorithm proposed by Kenneth O. Stanley and Risto Miikkulainen [5]. In this section, we will first present some background information about neural networks, and then we will present our implementations of these neural networks

A. Background knowledge on neural networks

A neural network is composed of at least two layers, the input and the output layer. Connections between neurons, called weights, are used to link a node to another. Each weight and neuron contains a value. The values in the input layer are attributed by the environnement. In this context, the number of neuron in the first layer are the number of sensors present in the car. Their values depend on the values of the sensors. To compute the value at a certain neuron we do a sum of all the multiplication between each pair of neuron and weight. Then we apply an activation function to the result. This function is shown in (Eq.2). And the activation function is shown in (Eq.3). By applying the summation function (3), we can calculate the values of each of the neurons, going layer per layer. To get the output of the network, we check the values of each neurons in the last layer. In our context, the output of the network is used to know the speed that each wheel should goes at.

$$y = f(x,w) = \varphi\left(\sum_{i=1}^{m} w_i x_i\right) \quad (2)$$

$$\varphi(x) = \frac{1}{1+e^{-x}} \quad (3)$$

To define a neural network, we need two things: the topology (the number of layers and the number of neurons in each) and the value of the weights. This means that only these two parameters are needed to recreate the neural network, and thus the only parameters that affect the performance of a Neural Network. A random set of topology and weights is not going to perform well. Once trained, meaning finding the right weights and topology, a neural network will do a better prediction according to its inputs. The goal of our training algorithm is to find the best configuration with real-time generated data.

B. The Fully Connected (FUCONN) Algorithm

The FUCONN algorithm uses a neural network with a fixed topology, meaning that the number of layers and the number of neurons in each layer won't change as the simulation progress (Figure. 3). In the context of our platform, the input neurons each receives the value of one of the sensors. In our examples we use seven proximity sensors per cars meaning the neural network of the car has seven input nodes in the first layer. The two node in the last layer, after being calculated, correspond to the value that each wheels of the car should go. The number of hidden layers and the number of neurons in each one can be specified before launching the simulation. In our examples, we demonstrate the Fully Connected Algorithm with one hidden layer consisting of three neurons.

To implement the neural network, we must have a way to encode the neural network. The only parameter that define the fully connected neural network are the weights, as the topology is fixed. We can then represent the neural network with an array containing all the values of all the weight. The size of this array stays the same during the simulation, because of the fixed topology. This representation comes handy because we can now change the Neural Network simply by altering the array.

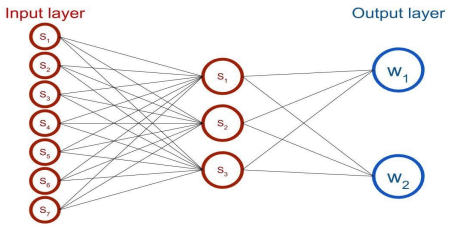

Figure 3. The topology of the FUCONN algorithm

In order to improve the fully connected algorithm's network the WENNG algorithm need to be able to produce a new FUCONN algorithms using two others; to redefine the crossOver() function. Therefore, the FUCONN algorithm needs to define what reproducing itself means. As the encoding of this algorithm is an array of weights, we need to find a way to merge the arrays of the two parents. The way NS WENNG does it is by randomly selecting the value of either one parent or the other at each position in the genotype. When the FUCONN is mutated, it picks a random weight and slightly change its value.

C. The NEAT algorithm

The fully connected algorithm has a fixed topology, meaning that it only searches solutions in its arrangement

of weights. However, the NEAT algorithm has a different approach by making the topology of the network variable, and thus searching in a wider range of possibilities. This wider range can be useful to find solution that otherwise could be hidden. The NEAT algorithm starts with only one connection going from a random input neuron to a random output neuron.

To be trained by the WENNG algorithms, it needs to define the crossOver() function. To do this, creates a new network with hidden neurons and connections from both of its parents. This function is explained in the traditional NEAT algorithm[5].

One of the differences with the Fully Connected Algorithm is within the mutations possibilities. The FUCONN algorithm can only (1) randomly changes weights. The NEAT algorithm can also (2) add a neuron between two already connected neurons or (3) connect randomly two random.

C. The Improved NEAT (IMNEAT) algorithm

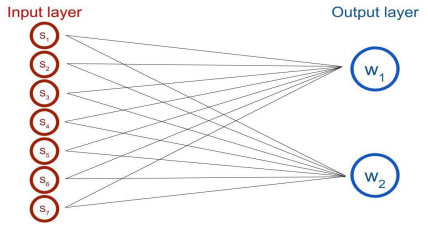

Figure 4. How the topology of the NEAT algorithm starts

In our demonstrations, we compare the traditional NEAT algorithm with IMNEAT. The difference between IMNEAT and NEAT resides in the number of starting connections . Through extensive simulation, we found that starting with only one random connection was not the best way in the context of autonomous vehicles. We noticed that it was useless to have some sensors doing nothing at the beginning. In our context it is logical that every sensor impact the decision of the speed of the wheels. To improve the NEAT algorithm we added some connections when the network is created (see Figure.4). When IMNEAT starts, it has all its neuron connected.

V. Simulations Results

In this section, we conduct a simulation study using the new version of the LAOP platform shared in [4] to evaluate and compare the performance of our proposed scheme, NS WENNG and GRD WENNG, training NEAT and FUCONN. We then compare our improved version of NEAT against the traditional NEAT. We evaluate several performance metrics based on the fitness : 1) the best car's fitness at each generation; 2) The average fitness at each generation 3) the rate at which the fitness increases as the generation increased and 4) the maximum fitness reached in the 20 generations. We recall that a generation is the process of simulating and then altering the set of cars. The fitness is a value assigned to each car at the end of each generation to determine how the car is performing.

A. Simulation Configurations

In the first scenario (1), We ran extensive simulations and multiple variation of the ***WENNG*** algorithm with several parameters as shown in Table.I.

Table I. Parameters of Scenario 1 simulation

Parameter	**Value**
Number of simulations	5
Number of generations	20
Number of sensors	7
Maximum generation time	60 seconds
Car density	50

In the second scenario (2), we run multiple simulation of the NEAT algorithm and the IMNEAT algorithm to find which one is performing better. All parameters for this scenario, except number of simulation stays the same. The number of simulation is set at 10.

B. Results Analysis

Scenario 1 : Variations of the WENNG algorithm

We present the comparison between NS WENNG and GRD WENNG learning approaches on the NEAT algorithm and the FUCONN algorithm.

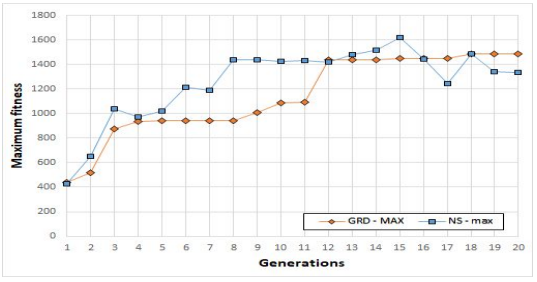

Figure. 5. Comparison of the best performing cars of **NS WENNG** vs **GRD WENNG** at each generation on **NEAT**

The Figure 5 represents a comparison of the best performing car at each generation using the *NEAT* algorithm. Here, we can see that the *NS WENNG* algorithm is overperforming the *GRD WENNG* on the first 12 generations. Passing the 15th generation, the *NS WENNG* algorithm starts to underperform. This is probably due to the fact that in the NS WENNG algorithm, all the cars (even the best-performing ones) have a chance to mutate. This is not the case in the GRD WENNG algorithm. If a good-performing car gets mutated, it has a chance to be less performant than before the mutation, thus resulting in a lost of performance. This is probably the reason why the blue line is unstable at the end.

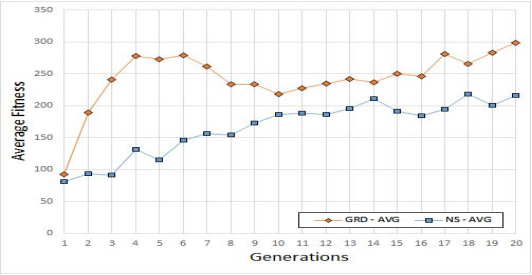

Figure. 6. The average fitness of **NS WENNG** vs. **GRD WENNG** at each generation on **NEAT**

Figure.6 shows a comparison of the average fitness between two algorithms. The GRD algorithm overperforms the NS WENNG algorithm during the whole simulations. This is surprising, because the NS WENNG performed better then the GRD WENNG in all the other cases. We believe that the high mutations rate of NEAT are behind this behaviour.

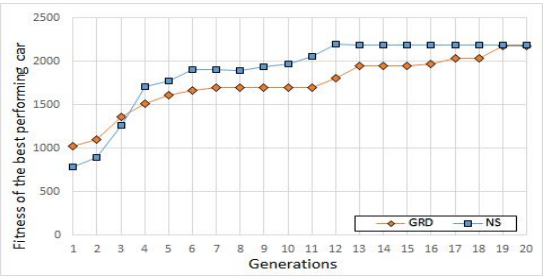

Figure 7. The best fitness at each generation between the **GRD** and **NS WENNG** using the **FUCONN** algorithm

In Figure. 7, the natural selection algorithm performed better than the GRD algorithm. Overall, the NS WENNG algorithm was faster than the GRD algorithm to get to the maximum fitness. With the FUCONN algorithm, it seems that the learning of the NS WENNG was more stable compared to the NEAT algorithm. This is probably due that the NEAT algorithm is more affected by the mutation and the reproduction, as it can completely change its topology. In the FUCONN algorithm, the mutations and reproductions are only affecting the weight's value. This means that if a good car is being mutated, it has less chance to completely change, and thus less chance of becoming less performant. We can conclude that the NS WENNG algorithm is better to train algorithms that are not affected a lot by mutations.

Figure.8 shows a comparison of the average fitness and the generation where the Natural Selection WENNG able to keep the average fitness at a high level. In almost all the cases, the NS WENNG overperforms the GRD WENNG algorithm.

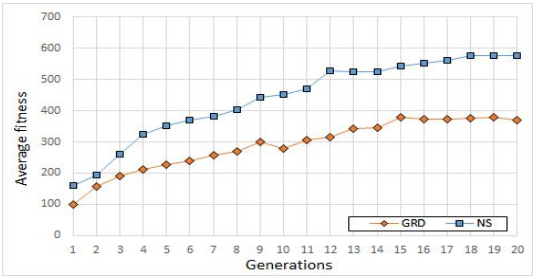

Figure. 8. The average fitness at each generation using the FUCONN algorithm between the GRN and NS WENNG

SENARIO 2: Comparison between our IMNEAT and the NEAT algorithm

In this scenario, we compare the IMNEAT algorithm and the NEAT algorithm. The difference of the two algorithms resides in the number of starting connections. The IMNEAT algorithm starts with all nodes connected. The NEAT algorithm starts with one connection between two random nodes. We are comparing them on the same four criteria explained at the start of this section.

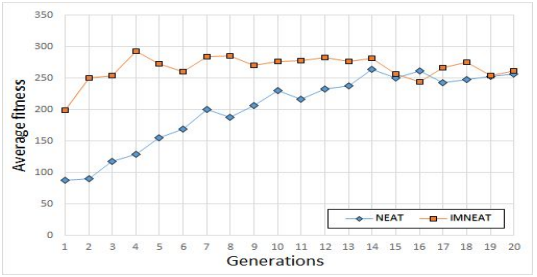

Figure.9. The average fitness of **IMNEAT** vs. **NEAT**

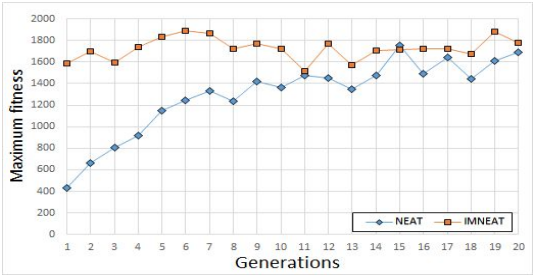

Figure.10. The best-performing cars of **IMNEAT** vs. **NEAT**

As we can see in Figures 9 and 10, the IMNEAT algorithm is starting with a *head start*. At generation 1 of both the average and the maximum, the IMNEAT's fitness is higher than in the NEAT's one. As the generations progress, the NEAT algorithm catches the IMNEAT's. But, this is caused by the fact that the max fitness (of ~1900) reached at generation 6 for the IMNEAT algorithm. The traditional NEAT algorithm gets to the maximum of ~1800 at generation 15. Our approach of pre-assigning connections makes IMNEAT twice as fast to react the maximum.

In summary, the simulation results confirm that well-trained Neural Network with Genetic principle can

efficiently satisfy our goal of getting a car to run without hitting walls.

VI. Conclusion and Future works

In this paper, we have proposed two Ways of Efficiently train Neural Networks with Genetics (WENNG). Our first approach (NS WENNG) tries to follows the principles of natural selection by determining the likelihood of keeping a car alive by chance, thus keeping a part of the less-performing population that could have interesting traits later on. The GRD WENNG gets rid of the less performing population and tries to only reproduce the better performing ones. In most cases, the NS WENNG approach gives better results. Except in the case of the average study of the NEAT algorithm. This exception is probably due to the fact that mutations have a big impact on the NEAT's best performing cars and that the selection algorithm mutates all the car by comparison with the GRD algorithm that only mutates the new cars.

Furthermore, we have concluded from our performance study that the regular NEAT enforced with our heuristic needs less generations to be as performant as the traditional NEAT algorithm. The fact of pre connecting the neural network gives it a *head start*, because the traditional NEAT algorithm will need to get to this connected state anyways during its training.

ACKNOWLEDGMENT

This research was financially supported by the "Fonds Québécois de la recherche sur la nature et les technologies (FRQNT)." We would like to thank Barrette Brisson, for her valuable comments.

References

[1] F. Petroski Such, et al., "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning", CoRR abs/1712.06567, 2017.

[2] Hbaieb, A., Rhaiem, and Chaari, L, "In-car Gateway Architecture for Intra and Inter-vehicular Networks", IEEE 14th IWCMC Conference (pp. 1489-1494), 2018.

[3] J. Rezgui, C. Bisaillon and L. Oest O'Leary, "Finding better learning algorithms for self-driving cars", IEEE ISNCC 2019 18-21 June, Turkey.

[4] LAOP by L. Oest O'Leary, C. Bisaillon and J. Rezgui on GitHub, https://github.com/lool01/LAOP, [last visit and update 24/04/2019].

[5] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies", Evolutionary Computation Journal, 10(2):99-127, 2002.

[6] M. Bojarski et al., "End to end learning for self-driving cars". arXiv preprint arXiv:1604.07316, 2016.

[7] D. A. Pomerleau. Alvinn, "An autonomous land vehicle in a neural network", Technical report, Carnegie Mellon University, Computer Science Department, 1989.

[8] D. Silver et al., "Mastering the game of Go without human knowledge", Nature, 550, 354, 2017.

[9] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning", preprint arXiv, Research by OpenAI, :1812.02341, 2018.

[10] S. Du, H. Guo and A. Simpson, "Self-Driving Car Steering Angle Prediction Based on Image Recognition", http://cs231n.stanford.edu/reports/2017/pdfs/626.pdf, 2017.

[11] C. J. C. H. Watkins. "Learning from delayed rewards". PhD thesis, University of Cambridge England, 1989.