Learning driving behaviors through Evolutionary Algorithms

Jacopo Donà, Giovanni Ambrosi, Riccardo Parola, Dung Van
Department of Information Engineering and Computer Science
University of Trento, Italy
{jacopo.dona, giovanni.ambrosi, riccardo.parola, dinhdung.van,} @studenti.unitn.it

I. Introduction

THIS paper aims to explore the use of evolutionary algorithms for learning driving behaviors in a simulated environment. We explore different approaches starting from a more standard one based on reinforcement learning and then exploring different powerful Evolutionary based ones.

A. The environment

The environment selected for this study is the "Highway" environment from the Gymnasium library [1]. This environment simulates a highway scenario, as depicted in fig. 1, where a green car drives on a highway alongside other randomly spawned blue cars (if RNG/seed is not fixed). The green car has access to five different types of actions: Lane Left, Idle, Lane Right, Slower, and Faster. These actions enable the agent to control the behavior of the car relative to the other cars in the environment.

B. The observations

The environment authors provide several types of observations for the agent to use. We found that for all of them, simply inputting the observation into the networks leads to poor results, due to the high input size and to the activation sparsity (at each time step, most of the nodes have 0 as input value). For these reasons, we decided to focus on one of them and apply pre-processing to ease the training phases. We used the `TimeToCollision` observation, which at each time iteration returns a $V \times L \times H$ array, where V is the speed the controlled vehicle can assume, L is the number of lanes and H is the horizon of the agent, which encodes how far in front the vehicle can see. In the default configuration, all cells have entry 0 values except the ones where there is a vehicle that is seen by the agent, which contains 1 instead.

To reduce the input size, we consider only the matrix of the current velocity of the vehicle. From there, we convert the observation into a LIDAR-type of observation space, where the agent is fed information regarding the distance of cars on his left lane, his right lane, and in front of it. If the car is on the leftmost, the value of the lane on the left is set to -1 (same for the right lane in the rightmost case).

C. The reward

The reward is based on the ability of the car to avoid collisions and go as faster as it can with a bonus if it drives in the rightmost lane. The reward at each time step is computed by the environment as follows:

$$\alpha \cdot \frac{v - v_{min}}{v_{max} - v_{min}} - b \cdot collision + 0.1 \cdot rightLane \quad (1)$$

where α is the weight for the speed in the final reward, b is the one for collision and 0.1 is the default reward for staying in the right-most lane, $collision$ and $rightLane$ are flags either 1 or 0. We found that in order to correct some of the negative behaviors that some of the solutions learned we needed to set this parameter as $\alpha = 1$ and $b = -5$

α encourages the agent to pick up speed while travelling and b penalizes crashes heavily.

II. Methodologies

A. Deep Q-Network

A Deep Q-Network (DQN) is a reinforcement learning algorithm where a network learns to produce an estimation of the Q-value of a state-action pair. The Q-learning algorithm updates the Q-values according to the following equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \quad (2)$$

where: α is the learning rate, controlling the step size of the updates. r is the immediate reward received after taking action a in state s. γ is the discount factor, balancing the importance of future rewards. s' is the resulting next state after taking action a in state s. a' is the action that maximizes the Q-value in the next state s'. To improve the learning phase of the network, we integrated a memory replay mechanism, where the agent stores its experiences, typically in the form of transitions, in a memory buffer called the replay memory. Each transition consists of the agent's current state, the action taken, the resulting next state, the reward received, and whether the episode has terminated. To maximize the exploration of the environment and encourage the agent in gathering experience, we implemented a ϵ-greedy strategy search, with a probability of selecting a random action that decreases over iterations

The Q-network is divided into 4 layers. The first layer contains a number of neurons equivalent to the observation size, the second and third layers are composed of 64 neurons, and the output layer contains a number of neurons equivalent to the number of actions, which in our case is 5. The observation is fed to the Q-Network, which produces an activation value for each of the actions and the agents select the action with the highest activation.

B. Genetic Algorithm

One way to train a neural network to perform a given task is through genetic algorithms. In this family of algorithms, at each generation a population of N individuals is generated, each holding a different genotype θ.

Once a population is generated, each individual is evaluated through a fitness function (in our case, by performing actions in the environment and collecting rewards) and the results are stored. The top m individuals are chosen as parents for the next population, meaning that each individual of the following generation will inherit the parents' weights, mixed through a crossover strategy. Additionally, each gene in θ can be subject to a mutation, this is done to favor the exploration of the solution space.

C. CMA-ES

The CMA-ES (Covariance Matrix Adaptation Evolution Strategy)[4] is an algorithm designed to integrate the standard Evolution Algorithm with a self-adaptation strategy for a more accurate solution sampling.

Rather than sampling solutions solely through mutation and crossover, CMA-ES maintains a multivariate normal distribution, also known as the search distribution. This distribution captures the characteristics of the current population of candidate solutions.

After computing the fitness of the individuals, the top μ individuals contribute to the adaptation of the search distribution parameters.

- The mean vector of the distribution is updated to move toward the direction of the selected individuals.
- The covariance matrix is adjusted based on the selected individuals' successes and failures.

Once the distribution parameters are updated, a new population of λ individuals are sampled from the search distribution, and the process is repeated. This approach improves the mutation strategy of the GA, as the distribution is able to understand the direction of most improvement and modify the mutation magnitude of the genes accordingly.

D. NEAT

NEAT is a genetic algorithm (GA) for the generation of evolving neural networks. It alters both the weighting parameters and structures of networks, attempting to find a balance between the fitness of evolved solutions and their diversity. It is based on applying three key techniques:

- Genetic evolution (cross-over and mutation);
- Speciation to preserve innovations;
- Complexifying, developing topologies incrementally from simple initial structures.

Each network is encoded in a genotype that describes weights (magnitude, nodes connected, activation or not of them) and nodes(activation function, bias)[5].

NEAT can be very difficult to tune to a specific task due to the high number of parameters that can be tweaked. In the evolution process, it can prune or perturb weights and move nodes creating extra hidden layers and obtaining different topologies with very interesting shapes. We started from a fully connected single layer of neurons to prevent the network to learn to skip some of the inputs, this was an issue in the early stages of the training because it would only learn to slow down when there was a car in the same lane instead of surpassing. In the training processes, we faced different problems from the stochasticity of the initial networks and of the environment which would cause the network not to learn. To prevent this problem we developed an approach to use more than one environment at each generation. This allowed us to both generalize the results for multiple environments and avoid achieving a high fitness score due to a favorable traffic configuration of the random scenario, rather than the goodness of the genome.

E. Evolutional Algorithm for Neural Network Policy Interpretation

In this approach we combine a DNN with a Binary Decision Tree to create an interpretable policy. Decision trees are highly interpretable but lack intermediate representational power in reinforcement learning, while DNN can obtain strong results but are really complex to interpret. In our work, we address this by using a well-trained DNN to guide the decision tree.

For the experiment, we use KinematicObservation which provides relative position and speed information for each vehicle. The observation is represented as a $V \times F$ array, where V is the number of vehicles and F is the number of features (position and speed in x and y axes). The hierarchical structure of decision trees makes it difficult for Evolutionary Algorithms (EAs) to learn and prone to getting stuck in local optima due to the lack of intermediate representation capability. Uncontrolled vehicles' dynamic behavior, with randomly organized rows, limits the capability of the Binary Decision Tree. Additionally, when an unavailable action is taken, it is treated as an Idle action, introducing noise in decision patterns and affecting both reinforcement learning and supervised learning approaches.

In order to overcome the mentioned challenges, we proposed to train a Binary Decision Tree using guidance signals from a well-trained Deep Q-Network. The decision tree is expected not only to replicate the relationship between observations and actions but also capable to interpret the decision pattern of each action. To build the decision tree, it receives as guidance signals:

Q-value: Through the reinforcement learning process, we assume that a well-trained Deep Q-Network can learn the potential reward in each action (Q-value). These values measure the relative quality between actions, hence giving a continuous measurement space that benefits the fitness function in evaluating decision trees. *GuidedBackpropagation*: We employed the method [7] to measure the influence of input features on the output Deep Q-Network decision. This information identifies which features in which car mostly influence the decision, which can be used to evaluate the action similarity between decision tree and Deep Q-network. **Algorithm and Fitness function setup** We employed the evolutional algorithm setup mentioned in [6] but customize the grammar and fitness function. For the grammar, other

than the $DECISION$ node and $ACTION$ node, we added a $SELECTOR$ node that can select which uncontrolled vehicle to consider based on the condition. The $SELECTOR$ node will partially change the input of the decision tree, which is the position and speed of the controlled agent and selected uncontrolled vehicles. In this way, each uncontrolled vehicle is equivalence and the decision tree input is fixed in shape but can dynamically adapt to the environment. For the fitness function, we modified the conventional f-score metrics. Instead of calculating precision and recall based number of classifications, these metrics are calculated based on the action quality, considering both the potential reward and similarity to action produced by deep Q-network. The quality of decision tree action is measured in case of True Positive:

$$V(a\_dt) = Q(a\_dt) * sim(a\_dt) \tag{3}$$

Otherwise:

$$V(a\_dt) = (Q(a\_best) - Q(a\_dt)) * (1 - sim(a\_dt)) \tag{4}$$

In which, a_dt is the action derived from decision tree, $Q(a\_dt)$ is the Q-value of action a_dt derived from Q-network and $sim(a\_dt)$ is the output similarity between decision tree and Q-network.

By taking into consideration of potential reward and action similarity, this fitness function will not force the output of the decision tree to be exact as the best action of the reinforcement learning agent but give space to encourage the action that maximizes both the potential reward and action similarity.

III. Results

Before reporting results some clarification +s must be done. Each algorithm solution is evaluated both by its fitness scores and also through its behavior in the environment. There might be cases where an individual scores a high fitness but crashes at the end of the scene and individuals that reach lower fitness values but has better behavior in the environment (it turns to avoid collisions or slows down if does not have chances to change lane). We report both the actual score value of each algorithm's best solution through 100 simulations and describe the behavior we can note by looking at the simulations.

A. DQN

The DQN is trained on 4000 iterations, as it needs to gather experience to learn to estimate the $Q(s, a)$. For this algorithm, we left the environment seed able to change between each iteration, in order to increase the number of (s, a) pairs the model can evaluate, and the algorithm is evaluated only one run.

Once deployed, it has very robust results, with the car moving fast through traffic, trying to be in the right-most lane when possible, and rarely crashes.

B. Genetic Algorithm

Various runs with different combinations of cross-overs, mutation, and fitness computation were tested. For each generation, 100 individuals were generated as offspring and 20 were selected as parents.

We found the best results with rank-based parents selection, single-point genotype crossover, and 0.1 mutation probability. The mutation magnitude is sampled by a Gaussian, with $\mu = 0$ and $\sigma = 1$, and by evaluating the fitness of a solution on 3 different environments, whose seed remains fixed through the evolution. By doing so, we hope to achieve a trade-off between having a more generalized solution, as it needs to perform on 3 different environments, without having the fitness landscape variate at each iteration.

The genotype is represented by a neural network with a fixed topology consisting of 3 input nodes, 16 hidden nodes, and 5 output nodes. To evolve the individuals, the weights are extracted by the network, flattened in a list, and treated like a real-valued vector.

By looking at the simulations, we see that the GA learned to drive at maximum speed to achieve better rewards at each timestep, and it is capable of making turns to avoid cars but in situations where the car should make two turns in rapid succession, it struggles (example: the current lane and the one on the left are occupied). Also, it does not seem to ever want to slow down. In situations where all four lanes are occupied and the best approach would be to slow down and wait for one opening to occur, the agent just crashes into the back of one car.

As reported in the Table in the appendix, the GA is able to obtain good scores in runs where the traffic conditions are not too difficult, but the overall performances are very unstable.

We also tried a mutation-only evolutionary strategy using an adaptive mutation probability based on fitness [3], but it led to poor results and was not further investigated.

C. CMA-ES

For CMA-ES we used $\lambda = 100$, and tested multiple values of μ parameter, ultimately setting it to 20.

We also tested networks of different sizes but ended up using the same network of the Genetic Algorithm (3-16-5), and again the genotype consisted of the flattened vector of weights of the neural network, while the topology remain fixed. We found better results by initializing the mean vector to 0, rather than sampling a random one. Elitism was kept disabled as, per the authors of the library, it impaired the search process.

I report both runs with 3 evaluations with fixed seeds and with only 1 evaluation on changing seeds between generations, as they obtain results with different pros and cons.

CMA-ES with 3 seeds learns a behavior similar to GA, it moves at the maximum velocity and is slightly better than GA at making turns for avoiding cars, but still struggles when the traffic blocks all the lanes, and the best approach would be to slow down.

CMA-ES with 1 variable seed was evolved for 300 epochs in order to have the same number of "environment tries" the other variant has. It learns a more careful approach, as it picks

up speed when no cars are around and, if it is approaching a car in its lane, it slows down before making the turn. This agent is less prone to crashing compared to the previous one, but due to the average low speed, the reward at the end of the simulation is not very high. Also, the crashes significantly lower the mean fitness.

D. NEAT

After many different implementations and runs NEAT outperforms in the top cases all the other methods scoring second in the mean fitness to DQN. We have shown that as in fig. 6 the best number of environments to use during the training is 10, allowing to train in reasonable time and achieving a better score overall with respect to the 3 and 5. We set the number of generations to 100 and the population size equal to 50. Evolving the population in random environments guaranteed a final individual capable of achieving a top-10 score of 41.8 out of 44. The mean is lower (17.15) than DQN mainly because of the initialization of the environments. There are cases where the car spawns right behind two cars at timestep 0 and does not have enough time to turn, causing a collision and returning a negative value of the fitness. Moreover, in some scenes, the car tends to slow down in order to avoid collisions instead of overtaking the cars it encounters lowering the score.

E. Binary Decision Tree

The training process is performed on the records of the deep Q-network instead of the environment. In this way, the training speed can be boosted up in the trade-off with the environment exploration rate.

From the table of F-score in table II, we can easily observe the effect of noise between Idle and other actions on the performance of the decision tree, as some actions are taken as Idle instead of one defined in deep Q-network agent. However, in general, the decision tree has the capability to learn the dynamic observation with $SELECTOR$ operator that can match the output of deep Q-network. This result can be achieved by the high correlation between fitness function and original f-score, visualized in fig. 8.

During the training, we observed the sensitivity and difficulty to estimate the float condition for split node, which having huge effects on the result decision. This can be further improve by adding inner estimation loop for values estimation.

Algorithm	Mean $\pm$ std	Top 10% mean
DQN	**24.35**$\pm$14.39	41.29
GA	9.02$\pm$10.74	25.03
CMA-ES-1	10.24$\pm$9.10	26.0
CMA-ES-3	9.10$\pm$11.61	33.68
NEAT-3	12.15$\pm$13.67	39.82
NEAT-5	5.59 $\pm$8.10	23.82
NEAT-10	17.11$\pm$ **16.50**	**41.8**
Decision Tree	10.87 $\pm$ 6.87	26.32

TABLE I
MEAN STD AND TOP 10% MEAN OF DIFFERENT APPROACHES

IV. DIFFICULTIES

The environment is very slow to simulate, especially for evolution algorithms when individuals have to complete multiple runs on the environment when computing the fitness. For these reasons, we were forced in using a modest-sized population and run for a reduced number of generations.

The basic configuration of the reward system in the environment is very confusing. It's very easy to obtain good scores even if the car collides with the ongoing traffic, which we believe instead should be treated as a catastrophic outcome. We spent some time trying to figure out better parameters for the fitness rewards.

The environment by configuration does not punish the agent for taking illegal actions. For example, if the car is in the rightmost lane available and takes a turn right action, rather than crashing off-road the environment converts the action into an idle, maintaining the car on the current lane at the same speed. We believe this inhibits the agent learning and can ultimately bring confusion in how the agents learn the turn actions.

With our current observations, the agent is not able to recognize the velocity of other cars. In addition, when a car switches lanes it only recognizes its new position if it has fully occupied the new lane, the transition could therefore be misinterpreted and lead to a crash if the agent is near.

For NEAT we noticed that results depend also on the initial population. They could be improved by running the algorithm with different populations and averaging between the different solutions. Otherwise, researches on the best starting points could be conducted, helping the population to spawn directly in a favorite starting position.

V. CONCLUSION

Overall this is a very challenging environment for many of the methods that we used because of stochasticity and complexity. We were able to achieve good results with the DQN and NEAT. As mentioned, the environment takes time to simulate, and especially for evolutionary algorithms where each individual is tested on multiple evaluations, the time to execute grows exponentially. From the fitness trends, we observe that with Evolutionary approaches we do not reach convergence, meaning the results could further improve with more individuals/generations. In addition, the traffic configuration between different seeds are really different, where some environments can be solved really easily while other require multiple actions to avoid crashes. A possible improvement could be to implement a way of regularizing the difficulty of the environment, in order to maintain an adequate challenge (and possibly increase it) throughout the generations.

VI. CONTRIBUTIONS

- Jacopo Donà worked mainly on the Deep Q-Network, the Genetic Algorithm, and the CMA-ES
- Giovanni Ambrosi worked mainly on NEAT development
- Riccardo Parola worked mainly on NEAT development
- Dung Van worked mainly on Binary Decision Tree

REFERENCES

[1] E. Leurent, "An Environment for Autonomous Driving Decision-Making", GitHub repository, 2018 (highway-env.farama.org/)

[2] K.O. Stanley, R. Miikkulainen "Efficient evolution of neural network topologies", Proceedings of the 2002 Congress on Evolutionary Computation

[3] S. Marsili Libelli, P. Alba "Adaptive mutation in genetic algorithms", Soft Computing 2000

[4] N. Hansen "The CMA Evolution Strategy: A Tutorial", 2016

[5] Hunter Heidenreich "NEAT: An Awesome Approach to NeuroEvolution", 2019

[6] Custode, Leonardo L. and Iacca, Giovanni "Evolutionary Learning of Interpretable Decision Trees", 2023

[7] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller "Striving for Simplicity: The All Convolutional Net", 2015

APPENDIX A
FIGURES AND TABLES

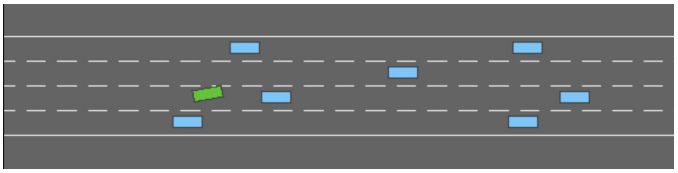

Fig. 1. Highway Environment

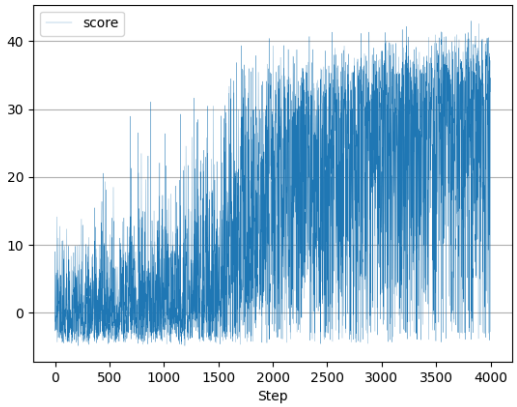

Fig. 2. DQN training scores

T/P	Left	Idle	Right	Faster	Slower
Left	**374**	71	10	32	13
Idle	11	**213**	26	108	142
Right	66	16	**333**	58	27
Faster	50	1	20	**392**	37
Slower	46	2	1	101	**350**

TABLE II
CONFUSION MATRIX OF THE BEST BINARY DECISION TREE

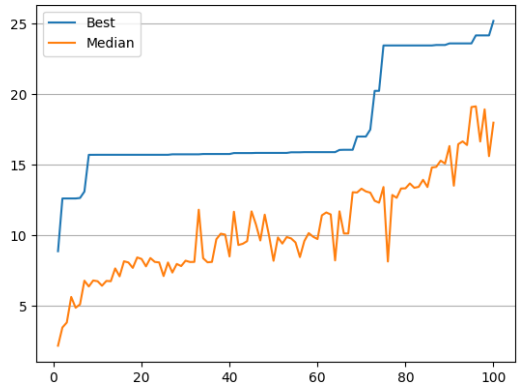

Fig. 3. GA fitness trend on 3 environments over 100 generations

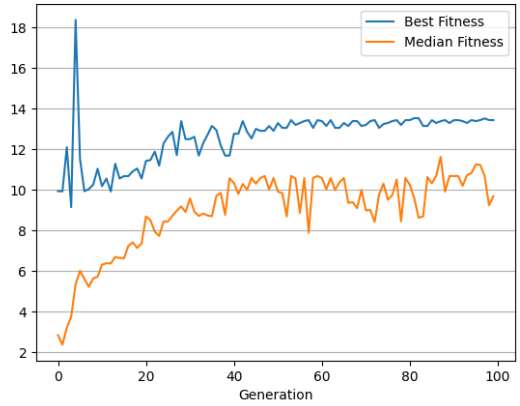

Fig. 4. CMA-ES fitness trend on 3 environments over 100 generations

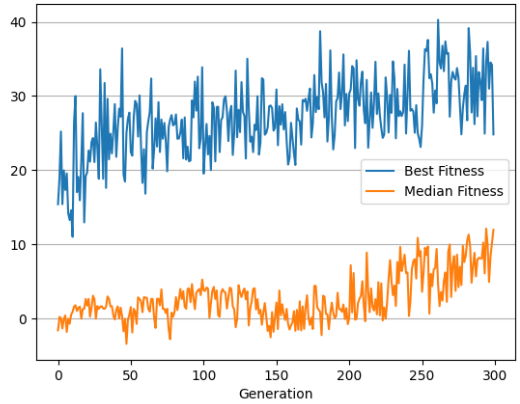

Fig. 5. CMA-ES fitness trend on 1 variable environment over 300 generations

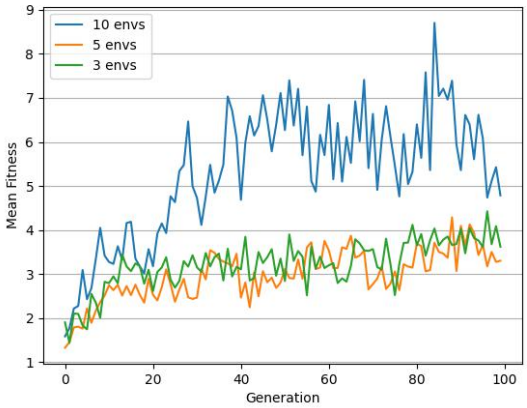

Fig. 6. NEAT mean fitness trend on different environments over 100 generations for each number of the environment set

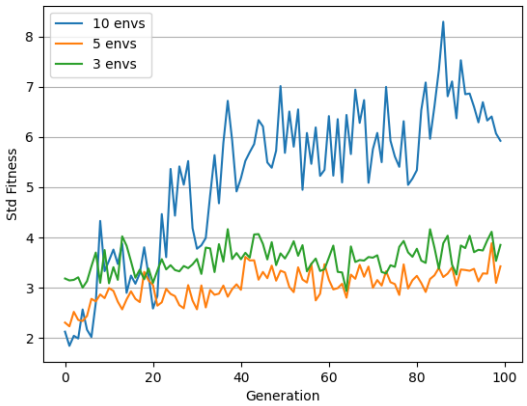

Fig. 7. NEAT standard deviation trend on different environments over 100 generations for each number of the environment set

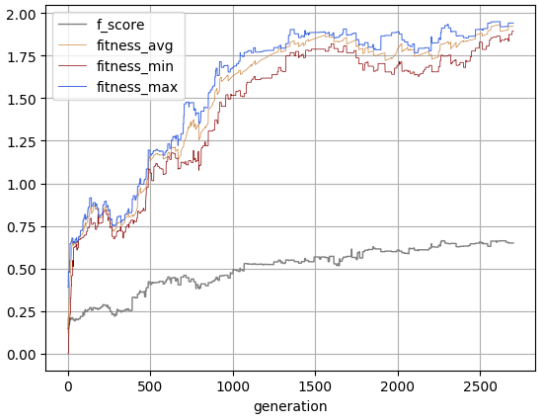

Fig. 8. Binary Decision Tree f-score and fitness related to tree quality