# Comparative Study of Neuro-Evolution Algorithms in Reinforcement Learning for Self-Driving Cars

**Ahmed AbuZekry[1], Ibrahim Sobh[2], El-Sayed Hemayed[3], Mayada Hadhoud[4] and Magda Fayek[5]**

*[1] Senior Algorithm Software Engineer, Valeo, Cairo, Egypt*
*[2] Senior Chief Engineer, Valeo, Cairo, Egypt*
*[3]Zewail City of Science and Technology, Egypt*
*[4] Assistant Professor at Cairo University, Cairo, Egypt*
*[5] Professor at Cairo University, Cairo, Egypt*

## ABSTRACT

Neuroevolution has been used to train neural networks for challenging deep Reinforcement Learning (RL) problems like Atari, image hard maze, and humanoid locomotion. The performance is comparable to the performance of neural networks trained by algorithms like Q-learning and policy gradients. This work conducts a detailed comparative study of using neuroevolution algorithms in solving the self-driving car problem. Different neuroevolution algorithms are used to train deep neural networks to predict the steering angle of a car in a simulated environment. Neuroevolution algorithms are compared to the Double Deep Q-Learning (DDQN) algorithm. Based on the experimental results, the neuroevolution algorithms show better performance than DDQN algorithm. The Evolutionary Strategies (ES) algorithm outperforms the rest in accuracy in driving in the middle of the lane, with the best average result of 97.13%. Moreover, the Random Search (RS) algorithm outperforms the rest in terms of driving the longest while keeping close to the middle of the lane, with the best average result of 403.54m. These results confirm that the entire family of genetic and evolutionary algorithms with all their performance optimization techniques, are available to train and develop self-driving cars.
**Keywords:** autonomous vehicles; computer vision; deep neural networks; evolutionary strategies; Neuroevolution algorithms

## Introduction

Autonomous vehicles are an important technology that will help improve the quality of living in many aspects. Most accidents occur due to some sort of human error, so autonomous vehicles are expected to help eliminate this error, thus dramatically decrease the number of accidents and fatalities. Moreover, improve the mobility of those who are unable to drive for whatever reason. Furthermore, make better use of vehicles, as most vehicles are driven to work, then stay parked through a significant portion of the day. Additionally, help people use their time more efficiently, as they can be more productive in their commute time, like reply to mails, read books, prepare for meetings, or at least rest.

Therefore, Self-driving cars are one of the most active research topics in the past few years. Researchers are trying to make cars take decisions on its own, without driver's intervention, and only based on the input sensor data. The self-driving car pipeline mainly consists of environment perception, environment mapping, motion planning, controlling. Some of the conducted research improves the performance of one of the stages in the pipeline, like improving object detection which is part of the environment perception. Another approach is to improve the system end-to-end, which is what this work does.

## Related Work

There are several approaches to this task, and one of them is to train a neural network to take the decision end-to-end. In 1989, [Pomerleau, 1989] successfully trained a 3-layer back-propagation neural network to follow the road under certain circumstances. The network was trained on 1200 simulated road snapshots, and tested on real roads. The network's input is camera and laser range finder, and the output is the direction the vehicle should travel to follow the road. The network was tested on Carnegie Mellon University's (CMU) autonomous test vehicle, NAVLAB. NAVLAB was successfully navigated through a 400m path in a wooded area in CMU with a speed of 0:5m=s in a sunny fall conditions. Behavioral Cloning is one of the approaches used in training neural networks for autonomous driving. In behavioral cloning, the network is trained to imitate the behavior of the driver. First a user drives around and records data, then the network is trained on this data. [Bojarski et al., 2016] were able to train a neural network to predict the steering angle from the raw pixels from a single front-facing camera. Although the main focus of the paper was to prove that the end-to-end systems are better at learning and optimizing the internal representations of the necessary processing steps than explicitly decomposed systems, but a good performance at driving the car was achieved. This leads to better performance and smaller systems, because the internal components self-optimize to achieve overall maximum performance, and the system is smaller, because it is a single network. The training data was collected from driving around in New Jersey, Illinois, Michigan, Pennsylvania, and New York in a diverse set of lighting and weather conditions. A mode of training is chosen before training the network, either staying in lane, switching lanes, etc. According to that choice, a subset of the training data is chosen, and data augmentation for generalization is done. After training, the network is tested on a simulator first, for safety, before taking it on a road test.

Another approach to train neural networks in autonomous driving is reinforcement learning, where the agent, i.e the network, learns from a reward system from a live direct interaction with the environment. In [Mnih et al., 2016], four asynchronous methods were developed; Asynchronous onestep Sarsa, Asynchronous n-step Q-learning, Asynchronous advantage actor-critic, and Advantage One-step Q-learning. These methods were used to train a neural network to drive in The Open Racing Car Simulator (TORCS) [Wymann et al., 2000]. The agents interacted with the environment, and was fed the front RGB image of the current frame and it received a reward proportional to the speed of the car along the center of the track. There were four types of experiments; the agent controlling a slow car and a fast car, with and without opponent bots. The Asynchronous advantage actor-critic agent always outperformed the rest of the algorithms. In [Wang et al., 2018], Deep Deterministic Policy Gradient (DDPG) is adopted to train a neural network to drive around in TORCS. The reward is designed to encourage speed along the track axis, and punish deviation and vertical movement from the track axis. The model was trained for 200 episodes with no opponents and tested with 9 opponents in the track. It was discovered that the mean speed of the car and the average gain increased as training goes. Also, the total reward per episode and total distance covered increased with each episode. Moreover, the variance of the distance to the center of track decreased by training, and that means that the agent learned to drive properly and in a stable manner in the middle of the lane. In December 2017, Uber published an article1 with 5 papers published on the topic of neuroevolution [Conti et al., 2018], [Such et al., 2017], [Lehman et al., 2018b], [Zhang et al., 2017], [Lehman et al., 2018a]. Algorithms that were inspired by natural evolution strategies were developed and tested on Atari games, humanoid locomotion and image hard maze, which are all reinforcement learning tasks. The performance of those algorithms were compared to Deep Q-Networks (DQN) [Mnih et al., 2013] and Actor Advantage Critic (A2C), the synchronous simpler implementation of Asynchronous Actor-Critic Agents (A3C) [Mnih et al., 2016], and the neuroevolution algorithms outperformed the traditional back-prop algorithms in training speed and overall agent performance and cumulative reward. If the performance of those algorithms are good in reinforcement learning tasks, accordingly, in this work, neuroevolution algorithms are evaluated for self-driving cars.

The work presented in this paper is based on the papers published by Uber ai labs [Such et al., 2017], [Conti et al., 2018], where it has been proven that using genetic algorithms to train deep neural networks for reinforcement learning tasks yields better results than gradient-based learning algorithms in training time and agents cumulative reward. It also has been proven that using exploration algorithms, namely novelty search and quality diversity, improves the performance of ES algorithms in deceptive or sparse deep RL tasks. [Zhang et al., 2017] reduced the ambiguity of the relationship between ES and Stochastic Gradient Descent

[1] https://eng.uber.com/deep-neuroevolution

(SGD), and showed that with enough computation provided to approximate the gradient, ES sometimes outperforms SGD in deep RL. Furthermore, the relationship between ES and SGD was further explored in [Lehman et al., 2018a]. ES optimizes for the expected reward of a population of policies described by a probability distribution, instead of optimizing the reward of a single policy, so it explores different areas of the search space, and sometimes that improves performance, depending on the search space. Moreover, ES acquires robustness properties not attained by SGD. As in [Lehman et al., 2018b], gradient computation have been combined with neuroevolution to achieve two things; evolving deep neural networks (DNN) with over one hundred layers, which wasn't done before with neuroevolution, and adjusting the mutation to treat parameters with different sensitivities differently, solving major problems of random mutation in large networks. Most of the work done by Uber in the previously mentioned papers were mainly based on [Salimans et al., 2017], where it has been proven that ES is a viable solution strategy that scales well and is able to solve RL problems like MuJoCo and Atari faster than usual RL techniques.

The main contribution of our work is:

- Comparing the performance of different neuroevolution algorithms in the task of self-driving car.
- Comparing the neuroevolution algorithms with a competing RL algorithm, DDQN.
- Show that using neuroevolution algorithms in self-driving cars is competitive and can outperform traditional RL algorithms.

The paper goes as follows, related work is summarized, then the ES, GA, and RS algorithms are explained. These are the algorithms used in this paper for comparison against DDQN [Van Hasselt et al., 2016]. Moreover, the research framework, which consists of the simulation environment used, the data used by the network, and the network architecture, is explained in detail. Furthermore, the detailed conducted experiments are explained with the results, and Finally, the main findings of the paper are discussed.

## Evolutionary Methods

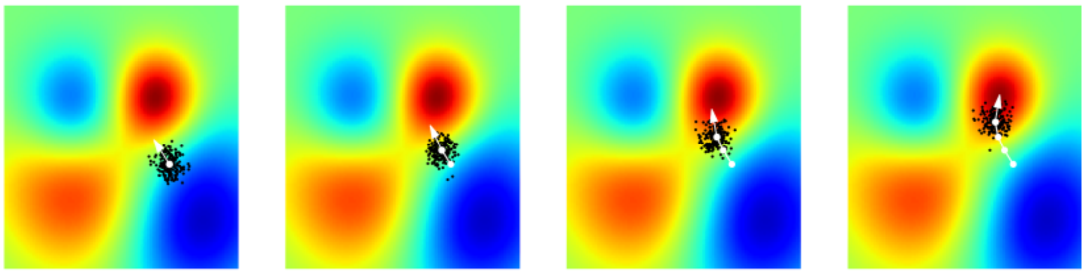
In the following, the 3 neuroevolution algorithms published by Uber are discussed

### 1 Evolutionary Strategy (ES)

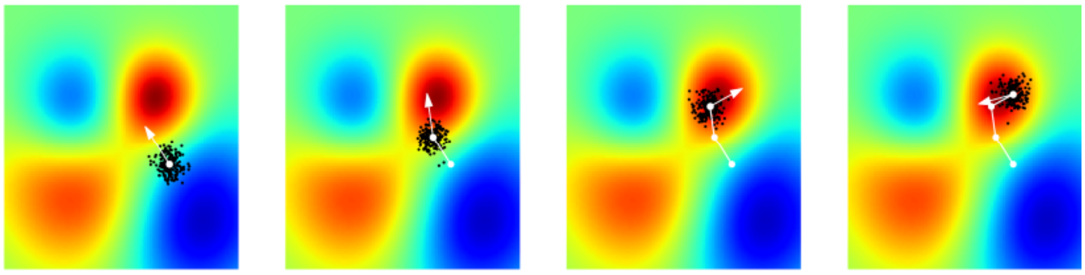
According to [Rechenberg, 1978], ES is a class of black box optimization algorithms inspired by natural evolution. From an abstract point of view, it can be seen as starting with a random parameter vector and repeatedly slightly manipulating that vector, and then moving that vector towards whatever changes worked best [Andrej Karpathy and Sidor, 2017]. In a more detailed description, the algorithm starts with a parameter vector, and then generates a population of N individuals by manipulating that parameter vector using a gaussian noise. Then the algorithm tests each individual in this population and accumulates rewards, and then changes the original parameter vector by a weighted sum of the population vectors. The highest performing individuals get higher weights. Accordingly, the algorithm tries to maximize the average performance of the population [Conti et al., 2018]. This code2 is developed to visualize a 2D vector space to illustrate how the algorithm works with different parameters settings. Fig. 1 illustrates different parameters settings.

ES has some advantages over regular reinforcement algorithms [Andrej Karpathy and Sidor, 2017]. It is highly parallelizable, as each individual can run in its own thread on the CPU, without the need for GPU. Also, there is no back-propagation, which makes ES much faster. Moreover, the algorithm trains by inserting noise into the parameters directly, so it is robust to parameter perturbations. These are just to name a few.

Figure 1: ES Algorithm Illustration

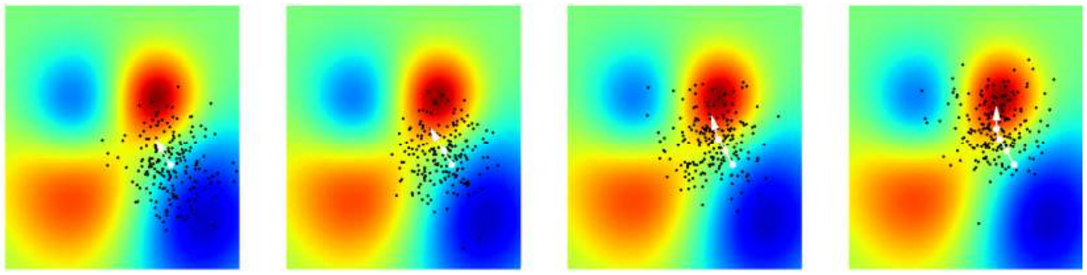


(a) ES mechanism demonstrated on a simple 2D toy example. It shows how the population is drawn around the mean parameter vector, and that mean parameter vector is moved towards the best performing individuals. The variance and the learning rate affects the algorithm performance.



(b) This example shows that the high learning rate affects the performance of the algorithm and makes it circle around the optimum position instead of directly settling there.

[2] https://github.com/karpathy/randomfun/blob/master/es.ipynb



(c) This example shows that the high variance in the population sampling makes the algorithm not know exactly where to go as the population is widely spread out and not converging.

## 2 Genetic Algorithm (GA)

Genetic Algorithms are also part of the black box optimization algorithms inspired by natural evolution. It produces a generation of N individuals, and each one of them is a neural network with it's own weights. At each iteration, the algorithm generates this population, and each individual runs an episode in the environment and returns a cumulative reward. The algorithm variant in [Such et al., 2017], which is also the one implemented in this work, does a truncation selection, where the top few performers of the generation are selected to be parents for the next generation. Then the process of generation production starts by selecting a parent from the top few performers from the last iteration at random, and for the populationsize - 1, a gaussian additive noise is added to the parameters of that parent, producing a new individual. The last individual of that generation is an unmodified copy of the best performing individual from the previous generation. The algorithm operates for a certain number of iterations or until a certain reward is achieved. In [Such et al., 2017], an innovative solution to store network individuals to save memory and network usage for distributed execution was implemented, but it wasn't used in this work due to limited availability of resources.

## 3 Random Search (RS)

In RS, a gaussian distribution is used to randomly sample individuals in a population and evaluate their fitness from the reward obtained from running an episode in the environment. The best performing individual in the population is chosen to be the product of the current iteration, but this individual is not used to generate the next generation. The next generation is produced in the same manner and the process continues for a certain number of iterations or until a certain performance is found [Stanley and Clune, 2017].

Comparisons of these algorithms are conducted with one of the most popular reinforcement learning algorithms, DDQN, to know how the algorithms compare to each other, and also to widely used regular RL algorithms.

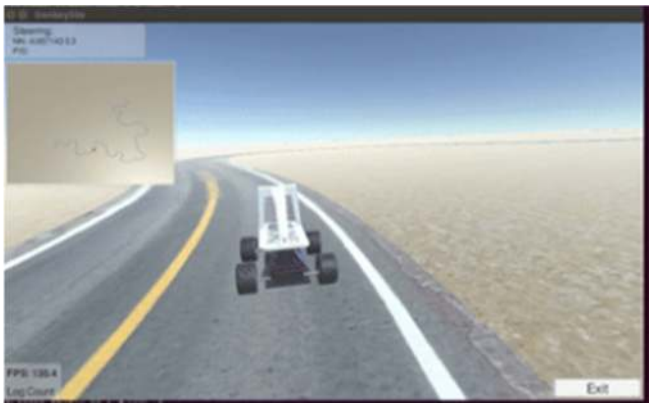
## Framework

### 1 Environment

### 1.1 The Simulator

A simple Unity based Donkey Car Simulator, which has the same interface as OpenAI gym environment, is used and extended to train reinforcement algorithms [Yu, 2018]. It is very convenient since the code developed by Uber AI Labs for [Conti et al., 2018], [Such et al., 2017], which is used for this work, was developed on OpenAI gym environment. So it is convenient in integration and good for proving the theory, before moving on to a more complex and urban environment. The environment is a highway with different curves, and some of them are challenging, with no other moving objects in the scene. The Donkey Car is always driving on the right lane and the goal is to drive in the middle of the lane. Fig. 2 describes the scene.

*Figure 2: DonkeyRL Simulator*



### 1.2 The Reward Function

[Yu, 2018] also developed a training code for a DDQN model that trained the model on the environment, where the model is trained to predict the steering angle of the car, and the throttle is kept constant. Two reward functions are proposed to train the model. The first one is as follows:

$$Reward = 1 - \frac{CTE}{CTE_{Max}} \tag{1}$$

The CTE term is the Cross Track Error, which represents how much the car is away from the middle of the lane, so if the car is exactly in the middle of the lane then this term is zero and it increases as the car drifts away from the middle of the lane. And the CTEMax term is an upper limit on the CTE, to control the maximum allowable drift from the middle of the lane. From this equation, if the car is in the middle of the lane, i.e. the fraction term is 0, then the reward is 1, and as the CTE term increases, the reward decreases to a minimum of 0. This equation drives the model to learn to drive in the middle of the lane well, but it has no punishment on the car drifting away, because it still gets a small positive reward. The other reward function is:

$$Reward = CTE_{Prev} - CTE \tag{2}$$

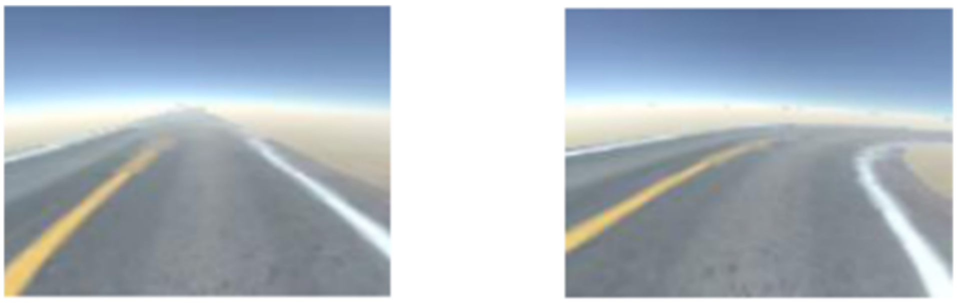
The term CTEPrev refers to the CTE value from the previous frame, and CTE is the CTE value from the current frame. Eq. 2 would give a positive reward if the agent was drifting from the middle of the lane in the previous frame and in the current frame it came closer. At the same time it would punish and give a negative reward if the agent was close to the middle of the lane and it drifted away, and this is a better way to guide the model to learn not to drift. Nevertheless, Eq. 2 rewards a zero value if the agent drives at the same distance from the middle of the lane in consecutive frames. It is the same value the agent is rewarded, whether this distance is exactly in the middle of the lane or away from it, so it the reward value is not indicative of whether the agent is doing a good job or not. The design of the road is randomly regenerated with each new episode, which helps randomize the input data for the model to not over-fit on a sequence of frames, and to generalize better.

## 2 Data

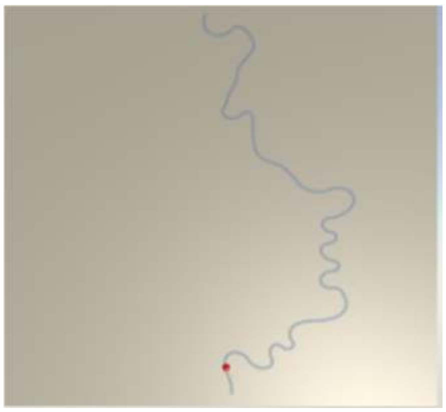
The input data comes from the front camera sensor, which is an RGB image of (120 x 160 x 3) pixels. Fig. 3 shows examples of different parts of the road.

No preprocessing is done for the image, except for resizing it to (80  80  3). Frame skipping is conducted to speed up training, but it was not used with our networks to not risk missing important frames. Also, the image is turned to gray scale, and edge and line detection are used to remove background noise, but in this work. Color images are used to make benefit of the added information, and the network would learn to discard background noise by itself, so the raw RGB image was fed to the network.

Figure 3: Environment Road



(a)Straight Road (b) Curved Road

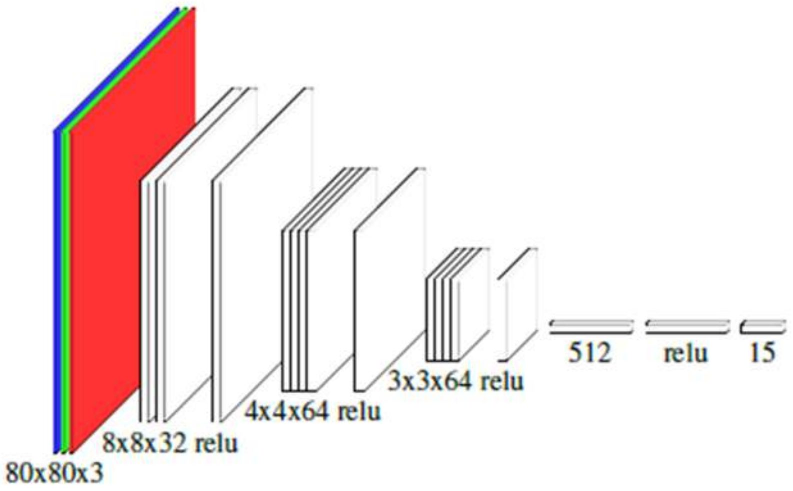


(c) Road Bird-eye View

## 3 Network architecture

The network architecture is shown in Fig. 4 and is described in detail along with the applied modifications. The input image to the network is an RGB image (80 x 80 x 3), while [Yu, 2018] is a gray-scale frame stacked (80 x 80 x 4). The first hidden layer convolves 32 8 x 8 filters with stride 4 with the input image and then applies a rectifier nonlinearity [Jarrett et al., 2009], [Nair and Hinton, 2010]. The second hidden layer convolves 64 4 x 4 filters with a stride of 2, and again apply a rectifier non-linearity. Then the third hidden layer convolves 64 3 x 3 filters with a stride of 1, and then apply a rectifier non-linearity. Then there is a fully connected layer of 512 units followed by a rectifier nonlinearity. Then the output layer is a fully connected layer of 15 linear units. The output layer is 15 units, because the steering in the environment is a continuous value between -1 and 1 and the network can only handle discrete values, so the value is discretized into 15 values. Adam is used following [Yu, 2018].

Figure 4: Network Architecture



## Experiments

Before going into the details of the experiments, the evaluation function used to asses the performance of the algorithms is explained. The maximum reward from Eq. 1 is 1, and the minimum is 0 and the cumulative reward of the episode is the summation of the reward at each time step. Also, the total length of the episode is the number of frames of the episode. Eq. 3 is then formulated.

$$MLHP = \frac{\Sigma\ Reward}{TotalFrameCount} \tag{3}$$

Where the MLHP is the Mid-Lane Hit Percentage. By dividing the sum of the reward by the frame count, the percentage of time that the agent drives in the middle of the lane during the episode is calculated. The higher that number the better, which represents that the agent does not deviate from the middle of the lane. Also, the environment has access to the 3D Cartesian coordinates of the car, which can be used to calculate the distance covered by the car from one frame to another by the 3D cartesian equation, Eq. 4.

$$Distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{4}$$

By summing this term in every frame, the total distance covered in the episode is calculated. The ultimate goal is the car moving forward and covering a large distance while staying in the middle of the lane. The evaluation function is then formulated as follows:

$$Eval = Total\ Distance\ Covered * MLHP \tag{5}$$

If the car moves in a straight line in the middle of the lane and does not pass the curves, and if it steers hard to stay in the middle of the lane so it moves forward slowly, the final result will be a small number because of the distance term. Moreover, if the car moves away from the middle of the lane, the MLHP term will be small so it will make the final result small. The ultimate goal is for the car to move in the middle of the lane with minimum steering so it can cover the largest distance possible, while surviving curves. The same number of iterations and the same network architecture are used across the experiments for fair comparisons. For both the neuroevolution algorithms and the DDQN, the models are trained for 100 episodes, and tested for 20 episodes, and the average and best MLHP, and average and best final evaluation value are calculated. For the evolutionary algorithms, there is a gaussian noise used with the network parameters and the variance of this noise affects the results. So neuroevolution algorithms were trained with 3 different variances for the noise. From the results of Table I, almost all neuroevolution algorithms outperform DDQN in all aspects. This is not the case using Eq. 2 in Table II , where DDQN has some edge over ES and GA in Avg:Eval and BestEval. ES and GA performance using Eq. 1 is much different from Eq. 2 in terms of the distance covered by the car, and similar in terms of MLHP.

This difference in performance between Eq. 1 and 2 suggests that Eq. 1 is more informative to neuroevolution algorithms and guide them better through the search space to better performing areas. The performance of RS is similar using the two functions, because it is a purely random method that is not driven by any rewarding system. The results show that neuroevolution algorithms always produces models that can drive better around the center of the lane regardless of using which reward function. The difference in Eval is substantial using Eq. 1, which is partly due to MLHP, but it also signals something about the distance traveled. It shows that neuroevolution algorithms can produce models that can survive more turns on the road than DDQN.

Table 1: Reporting Avg. MLHP, Best MLHP, Avg. Eval, and Best Eval using Eq. 1 in training and testing

| Algorithm | Avg. MLHP | Best MLHP | Avg. Eval | Best Eval |
|---|---|---|---|---|
| DDQN | 81.89 | 90.18 | 141.23 | 448.89 |
| ES | 97.13 | 98.35 | 154.87 | 365.13 |
| GA | 94.79 | 95.66 | 358.47 | 910.02 |
| RS | 95.93 | 97.87 | 403.54 | 592.89 |

Table 2: Reporting Avg. MLHP, Best MLHP, Avg. Eval, and Best Eval using Eq. 2 in training and Eq. 1 in testing

| Algorithm | Avg. MLHP | Best MLHP | Avg. Eval | Best Eval |
|---|---|---|---|---|
| DDQN | 75.3 | 83.16 | 242.006 | 436.88 |
| ES | 93.84 | 97.29 | 107.96 | 357.55 |
| GA | 86.74 | 97.14 | 100.55 | 301.52 |
| RS | 94.55 | 96.83 | 223.2 | 719.86 |

## Discussion

Results of the neuroevolution algorithms, especially ES and GA, suggest that densely sampling in the neighborhood of the mean, is sufficient in some cases to find better solutions than gradient-based methods. Also, sampling around a good solution is sufficient to find even better solutions. In this work, the simplest implementation of ES and GA is used. A simple optimization function of ES is used, that is why weak individuals in the population could pull the step taken in the next iteration to worse places, hence not guaranteeing a better solution from one iteration to the other. ES optimization function needs some improvements, but still with this simple implementation, ES outperformed DDQN, which suggests a great potential for ES in the self-driving car task. As for the GA, there exists some techniques that would improve the GA performance [Eiben et al., 2003], [Haupt and Haupt, 2004]. Also, some crossover techniques like [Holland, 1992], [Deb and Myburgh, 2016]. Furthermore, indirect encoding as in [Stanley, 2007], [Stanley et al., 2009], [Clune et al., 2011]. Also, [Mouret and Clune, 2015], [Pugh et al., 2016] for quality diversity, and many other techniques and different approaches to genetic algorithms in general that improve the performance of the algorithm. The RS performance does not change much because it is a purely random algorithm that is not guided by any rewarding system, so it sometimes does not reach any viable solutions, but at the same time it sometimes reaches extremely high performing solutions. As the algorithms incorporate noise for the parameter perturbation for sampling individuals, the variance of that noise has an effect on the performance of the algorithm. The higher the variance of the noise, the larger the radius of sampling, and if the radius is large, then the sampling is not done in the neighborhood anymore. The parameters mean can be in a good performance area in the search space, and with a large noise variance, an individual can be sampled from a distant bad performing area. If the variance is set correctly, then the sampling is done in the neighborhood of the mean, then if the mean is in a good performance area, then the algorithm will reach better solutions. It is interesting to see the performance improvement of neuroevolution algorithms in self-driving car when implementing any of the mentioned techniques, also in a different environment like an urban environment.

## Conclusion

This work compares three evolutionary algorithms, Evolutionary Strategy, Genetic Algorithm, and Random Search, to one of the most popular reinforcement learning algorithms, DDQN, and to each other, in the task of autonomous driving using only raw pixel images. Those evolutionary algorithms were used on other reinforcement learning tasks like Atari 2600 and humanoid locomotion, but here used for self-driving car task. The experimental results show that the evolutionary algorithms were able to outperform DDQN in driving in the middle of the lane, and in driving for a longer distance in the track, and going through more turns. This opens the door for evolutionary algorithms with all of its advanced optimization techniques to be used in the field of self-driving cars even on more complex scenarios and perform well, maybe reach state-of-the-art performance.

## References

[1] Andrej Karpathy, Tim Salimans, J. H. P. C. I. S. J. S. G. B. and Sidor, S. (2017). Evolution strategies as a scalable alternative to reinforcement learning.

[2] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[3] Clune, J., Stanley, K. O., Pennock, R. T., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. IEEE Transactions on Evolutionary Computation, 15(3):346–367.

[4] Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K., and Clune, J. (2018). Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty seeking agents. In Advances in Neural Information Processing Systems, pages 5032–5043.

[5] Deb, K. and Myburgh, C. (2016). Breaking the billion-variable barrier in real-world optimization using a customized evolutionary algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference 2016, pages 653–660. ACM.

[6] Eiben, A. E., Smith, J. E., et al. (2003). Introduction to evolutionary computing, volume 53. Springer.

[7] Haupt, R. L. and Haupt, S. E. (2004). Practical genetic algorithms. John Wiley & Sons.

[8] Holland, J. H. (1992). Genetic algorithms. Scientific american, 267(1):66–73.

[9] Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. (2009). What is the best multi-stage architecture for object recognition? In 2009 IEEE 12th International Conference on Computer Vision (ICCV), pages 2146–2153. IEEE.

[10] Lehman, J., Chen, J., Clune, J., and Stanley, K. O. (2018a). Es is more than just a traditional finite-difference approximator. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 450–457. ACM.

[11] Lehman, J., Chen, J., Clune, J., and Stanley, K. O. (2018b). Safe mutations for deep and recurrent neural networks through output gradients. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 117–124. ACM.

[12] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937.

[13] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[14] Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. arXiv preprint arXiv:1504.04909.

[15] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10), pages 807–814.

[16] Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In Advances in neural information processing systems, pages 305–313.

[17] Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. Frontiers in Robotics and AI, 3:40.

[18] Rechenberg, I. (1978). Evolutionsstrategien. In Simulationsmethoden in der Medizin und Biologie, pages 83–114. Springer.

[19] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864.

[20] Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. Genetic programming and evolvable machines, 8(2):131–162.

[21] Stanley, K. O. and Clune, J. (2017). Welcoming the era of deep neuroevolution.

[22] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. Artificial life, 15(2):185–212.

[23] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567.

[24] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In Thirtieth AAAI Conference on Artificial Intelligence.

[25] Wang, S., Jia, D., and Weng, X. (2018). Deep reinforcement learning for autonomous driving. arXiv preprint arXiv:1811.11329.

[26] Wymann, B., Espi´e, E., Guionneau, C., Dimitrakakis, C., Coulom, R., and Sumner, A. (2000). Torcs, the open racing car simulator. Software available at http://torcs. sourceforge. net, 4(6).

[27] Yu, F. (2018). Train donkey car in unity simulator with reinforcement learning.

[28] Zhang, X., Clune, J., and Stanley, K. O. (2017). On the relationship between the openai evolution strategy and stochastic gradi*ent descent*. arXiv preprint arXiv:1712.06564.