A platform for Sharing Artificial Intelligence Algorithms in Autonomous Driving

An overview of Enhanced LAOP

Jihene REZGUI, Clément BISAILLON, Léonard OEST O'LEARY
Laboratoire Recherche Informatique Maisonneuve (LRIMa)
Montreal, Canada
jrezgui@cmaisonneuve.qc.ca, bisaillon.clement@univ.teluq.ca, leonard.oest.oleary@umontreal.ca

Abstract- **To solve the Autonomous Vehicle (AV) problem, an artificial learning model that translates sensor data into controls must be built. This way, the vehicle can react to its changing environment. To the best of our knowledge, there are no platforms where researchers can both develop new machine learning models, train them and compare them directly to others. We believe that such a platform can greatly impact the field of artificial intelligence and AV. In this paper, we propose an enhanced version of the Learning Algorithm Optimization Platform LAOP, called LAOP 2.0. It helps researchers in the field of artificial intelligence develop their models by allowing them to easily test and compare them. We also introduce the Learning Algorithm Sharing Platform (LASP), which makes it easy to share deep learning algorithms. As a demonstration of the versatility of our platform, we compared two learning algorithms. The first one, Fully Connected Neural Network (FUCONN), uses reinforcement learning to train itself. The second one, Mimicking HUman Behaviour (MHUB), uses supervised learning to adjust its weights and learns from human input. We demonstrate through extensive simulations that FUCONN outperforms MHUB after being trained.**

Keywords- Deep learning, Open-source, Neural networks, Evolution, Self-driving cars, Natural selection, reinforcement learning, supervised learning;

I. Introduction

The problem we currently face in the field of Autonomous Vehicles (AV) is the lack of an uniformed platform offering an all-in-one solution to test and compare different algorithms. Most algorithms are developed in their own environment. This makes it difficult to compare them because tests must be driven in the same environment to be relevant. Often, algorithms must be tweaked to fit the environment requirements and structure. This makes it harder to share algorithms with others and build upon the existing ones. We think that this slows down the research field and could be fixed by having a platform offering different tools to simulate, test and compare them.

Artificial Neural Networks (ANN) trying to solve the problem of AV does it in roughly the same way [1-2]. ANNs take as input values of sensors attached to the vehicle. Sensors can be cameras, distance sensors or other electronic devices that can transform real-world information in data. Note that a combination of these sensors can also be used to represent the environment [3]. Depending on sensors' values, the ANN outputs an action. This action is then performed on the car. With this structure in mind, it makes it possible to create a platform where different deep learning models can coexist.

We developed LAOP [4-6] to offer researchers a place to create, test and compare their learning algorithms. The open source code is available here [5]. If adopted, our platform could fix the problem we noted earlier. The renewed structure of LAOP focuses on leting researchers easily share their work and inspire themselves from the work of others. We changed the structure of the algorithms on the platform to allow more algorithms that are diverse. Algorithms conceived in the first version could only use the Way to train Efficiently Neural Network with Genetic principle (WENNG[6]). With the newer version, we also introduce Learning Algorithm Sharing Platform (LASP). LASP is an open-source online platform where researchers can share their algorithms developed on LAOP. This sharing platform is integrated directly into LAOP. This allows users to download new algorithms and compare against them easily.

Our contributions can be summarized as follows: (1) We demonstrated through extensive simulation how our platform could be used by comparing the FUCONN and the MHUB algorithms; (2) We improved the first version of LAOP to make it even easier for researchers to develop and share their deep learning algorithms; (3) We developed the LASP (Learning Algorithm Sharing Platform). LASP allows researchers to share deep learning algorithms; and (4) We propose an approach to compare performances of deep learning algorithms using different learning techniques.

The remainder of the paper is organized as follows. Section II provides a brief overview of related work. In this section, we provide a comparison between our tool and existing tools. Section III looks at the new version of LAOP by displaying the biggest changes and explaining how we handle the different kind of deep learning algorithms. In section IV, we showcase the algorithms we added to the platform. These algorithms allow researchers to have a base comparison to work with. Section V shows an overview of LASP, a platform that lets researchers easily share their algorithms. Section VI shows the results of our comparison between the FUCONN and the MHUB algorithms. Finally, conclusions are drawn in Section VII.

II. Comparison with other tools

OpenAI [7] has developed an open source tool called Gym[8] that allows users to easily simulate their

978-1-7281-5628-6/20/$31.00 ©2020 IEEE

reinforcement learning algorithm in diverse environments. This tool is great to develop and compare reinforcement-learning algorithms. However, the platform does not have an integrated way to share the algorithms researchers have been working on. In addition, it is specialized for algorithms using a reinforcement learning technique. Therefore, you cannot use this platform to compare algorithms that are using different learning techniques.

Amazon SageMaker[9] proposes a platform that allows developers to build, train and deploy machine learning models. This platform is primarily used to produce models for commercial purposes. It does not promote the sharing of the algorithms. Also, this platform is not open-sourced and is not free to use. Which does not encourage people to develop on the platform and thus does not help advance the research.

Udacity car learner simulator [10] is a driving simulator built to teach deep learning concepts to students. While the user drives the car, the platform records the value of sensors and the action performed. The user can then use deep learning algorithms to train on the generated data. The problem we found with this platform is that the learning and comparison processes are not directly integrated into it. It is solely used to generate the data.

To summarize, current available tools primarily lack a way to share their algorithm easily. People have to adapt their algorithms to each platform they use.

III. An overview of LAOP 2.0

We first introduced LAOP in [4] with the idea of helping researchers optimize and compare their deep learning algorithms. Comparing deep learning algorithms is really important, as it lets researchers see which performs best and improve on them. The first version of the platform was lacking certain features we think are indispensable. We built LAOP 2.0 with the idea of making the platform even easier to use, make the simulation process faster and make the sharing of algorithms on the platform a breeze.
The improvements we made to the first version can be summarized as follows: (a) we made the platform easier to use; (b) we changed the inputs and the outputs of the model; (c) we added the possibility to create multiple learning environments; (d) we added the possibility for algorithms developed on our platform to use any kind of learning technique; (e) we simplified the process of adding new learning algorithms ; (f) finally, we made it possible to download, directly from our platform, new algorithms shared by other researchers with LASP.

(a) Ease of use
It is important to make the platform as easy to use as possible. We simplified the way the parameters of the algorithms work. We also added more feedback from the simulation to ease debugging during the development of an algorithm.

We redesigned the way that the settings of algorithms work. The platform now creates a "settings scope" for the simulation in general (the global scope) and one for each of the tested algorithms. First, this structure makes it easier for algorithms to have their own settings since they only have to add them to their own scope without messing with the global settings. Second, this allows algorithms to overwrite settings that are in the global scope. For example, you could simulate multiple variations of the same algorithm but with different settings for the car density. This lets researchers easily compare the performance of each configuration.

We also wanted the new version of the platform to give the most feedback possible to the user. To achieve this, we implemented a console, we added a timeline and we added more information about the selected vehicle. The console allows the user to view key information about the running simulation. It also allows the user to go back in time and see how the cars moved in the simulation. This feature makes analysis of algorithms easier. Finally, we made it possible to view more information about the selected car. For example, we can now see the different values of each of its sensors at every time step.

(b) Movement of the vehicles
The cars in our simulations are controlled by sending them two values at every simulation step: acceleration (1) and steering wheel angle (2). The acceleration has values ranging between -1 and 1. A negative value is equivalent to activating the brakes and a positive one gives acceleration to the car. The steering wheel angle has a value ranging from -1 to 1. When it is set to zero, the wheels have no rotation. A negative value represents turning the steering wheel to the right and a positive value, the left. Those two values alter the forces that are applied to the car, thus changing its direction.

(c) The environnements
LAOP 2.0 lets users create flexible environnements. Environments are what defines the problematic that will be tackled. We were interested in the AV problem so we created an environment for this specific problem. Take note that our platform can be used to help research in other fields than autonomous vehicles. In our case, the environment is a randomly generated maze of obstacles where cars can drive. The environment structure is similar to the one used in gym[8]. Each environment has a *step* and a *reset* method that can be used to update the agents. An agent is an abstract concept representing an object controlled by the AI that interacts with its environment. In our examples, the agents are cars driving in the environment. To compare algorithms, the environment gives a fitness score to each car in the simulation according to the number of times the car hit a fitness wall. A car that hit a lot of fitness walls gets a better score than a car that does not hit any fitness walls.

(d) A better comparison method

One of the main goals of our platform is to easily compare multiple algorithms at the same time. The two metrics we considered are the learning speed and the ability to perform the task accurately. In this case, the task is "avoiding walls while driving". We limited ourselves to these metrics, as we cannot use metrics specific to a learning technique. For example, the error rate on models learning with supervised techniques cannot be used for comparison with models learning with reinforcement;

The way the environment attributes a score to a learning algorithm is inspired by the reward function in reinforcement learning technique. It works by attributing a score to the cars based on how good they performed. If the car performs well during a simulation, the learning algorithm gets a high score and vice-versa.

In our case, the environment used is a randomly generated maze. To generate the maze, we created a grid as shown in figure 1. At each edge on the grid, there can be two types of wall: a fitness wall or a real wall. In figure 1, the dark lines represent the walls that act like obstacles and the lighter lines represent the fitness walls. If the car touches a real wall, the car is considered dead and can no longer move. If the car touches a fitness wall, its score and lifespan increases. Each car has a timer decreasing at each frame called its lifespan. If the timer reaches 0, the car is considered dead and cannot move anymore. When the car touches a fitness wall, the lifespan of the car is extended and its fitness increases.

Figure 1. A maze generated by our platform

Each time a car touches a fitness wall, a predetermined amount of points are added to the fitness score. The equation of *points that a certain wall gives* (P) according to *the number of times that the car touched it* (t) is displayed in equation 1. We choose this fitness calculation over our last proposed method because it incites cars to travel further. Our previous method was only giving a score based on the distance between the car and the start, and the total distance traveled. This caused some problems: some very good cars got a bad score because they ended closer to the start.

The lifespan of a car is the amount of simulation frame left before the car is considered dead. A car starts with 100 frames. When a car crosses a fitness wall, its lifespan is extended by an amount computed from equation 2, where t is the amount of time the car crossed the same fitness wall.

$P = 10/t$	(1)
$T = 25/t$	(2)

In our environment, an episode is defined as the process of simulating cars controlled by the learning algorithm until all cars are dead. Cars can die by having no lifespan left or by hitting a real wall.

To extract the learning speed metric from an algorithm, we let the algorithm train for a predetermined number of episodes. Then, we perform the evaluation. The evaluation is the process of taking the best performing car of the last episode and evaluating its performance on 100 mazes. The performance of cars translates to the performance of the algorithm because the algorithm is controlling the car. The score of algorithms at that specific episode is the average of the fitness that the car got for each of the 100 mazes. Thus, when running multiple episodes we can track the performance of the learning algorithm over time.

To extract the final performance metric, we do a final evaluation process with the best performing cars of each algorithm, on a completely new set of 100 mazes.

The reason we evaluate on multiple mazes instead of doing it on just one is to reduce error related to randomness. To have a high score on one maze, the car needs to learn certain features of that environment. Features can be as simple as turning right or left or as complicated as doing a u-turn when in a dead end. Not all environments need the same features to be performed accurately and the features needed, in our case, depend only on the shape of the randomly generated maze. Therefore, some mazes can favorite certain cars that have learned the corresponding features. To represent each feature equally in our results, we test on multiple mazes and thus have a score that truly represents the performance of the car.

(e) New way of adding algorithms

In the first version of LAOP, researchers could only develop algorithms using WENNG. We thought that this was too restrictive. We changed the structure of algorithms on the platform to enable more types of learning algorithms. For example, our platform now supports algorithms that use supervised techniques for their learning process.

In LAOP 2.0, we simplified the way algorithms are added to the platform. Previously, we separated the learning process from the neural networks. In code, this was implemented by having to create two distinct classes that implemented each a different interface. We simplified this process by only needing to create one class that implements one interface. To make it even more simple, the interface to implement contains only two functions to redefine. This design choice gives more versatility to newly created algorithms, as they do not have to fit inside

a predefined model. Previously, we forced all algorithms to be trained a certain way. This is not the case anymore.

In the programming structure, we have one interface called the LearningAlgorithm that all algorithms must implement. This interface has two functions : *train()* and *test()*. Train is called when the simulator is in its training process. Inside the scope of this method, the instance of the current environment class is available. The function *test()* takes one parameter called the *Agent* and returns a CarController instance. The agent contains all the information of the sensors and the CarController instance contains all the information on how the car should move. The environment has multiple functions to control itself like step() and reset(). The CarController has a method called control() that takes as inputs the value of each sensor and outputs the action that the car then performs. This method is called at each step of the simulation to predict how the cars will move. This method defines how the learning is actually done. For example, FUCONN needs to evaluate the cars by giving them a fitness score, select the ones that will stay for the next generation and populate the array of cars by mixing the better ones.
Finally, there is a LearningEngine class that takes all the learning algorithms that will be tested and runs them one after the other, collecting data at each step of the simulation.

(f) Upload and download algorithm directly from the LAOP

Within LAOP 2.0, we have a utility that lets the user share their learning algorithms by compressing it into a jar file. Once the algorithm is in that format, it can be uploaded on our web application LASP where everybody can download it to compare it with their algorithm. The download and upload process can be done within the application.

IV. Proposed algorithms

We propose two deep learning algorithms with our platform that allows researchers to quickly compare them with theirs. One of the algorithms we proposed is the FUlly COnnected Neural Network (FUCONN) algorithm introduced in our past paper [6] and the other is a new algorithm that we introduced with LAOP 2.0. We called it the Mimicking HUman Behaviour (MHUB) algorithm. In this section, we will describe both of the ways that these algorithms learn and compare them briefly. MHUB learns from human controls and FUCONN learns from trial and error with genetics principles.

(a) Learning from human controls (MHUB)

MHUB is composed of an ANN that takes as inputs the values of each sensor the car has and outputs the actions that the car should do. The topology of the ANN is as follows: there are three deep layers, containing three, twelve and five neurons respectively. We decided this topology after trying different configurations.

To train the neural network, we let a human drive the car through the environment using her or his keyboard. While controlling the car, we collect data about the simulation. A dataset containing the values of each sensor and the action that the user did is generated. We can then train in real time the network using stochastic gradient descent (SGD) on the generated dataset. This way, the neural network will learn to mimic the way that the human drives in the simulation according to the values of the sensors. When the learning process is done, we let the car drive by itself by feeding in the values of the sensors in the ANN and getting an action from it similar to what a human would have done.

To implement SGD with the ANN, we used the library Deep Learning for java (DL4J) [11]. This open-source library allows us to easily create an ANN with a certain number of layers and neurons. It also allows us to make the network learn on our generated dataset. The activation function used in every layer was the hyperbolic tangent function. The loss function used is the squared loss and we used a Gaussian distribution as the weight initialisation.

(b) Learning with the genetic principles (FUCONN)

FUCONN also takes as inputs the value of each sensor of the car and outputs the action that the car should take. The difference between these two algorithms rely on the way they learn and the structure of their neural network.

The artificial neural network of FUCONN consists of two deep layers containing five and height neurons respectively.

This ANN is trained using the "Way to train Efficiently Neural Network with Genetic principle" (WENNG) algorithm proposed in our past work [6]. This algorithm works in three phases: the evaluation, the selection and the repopulation as shown in figure 2.

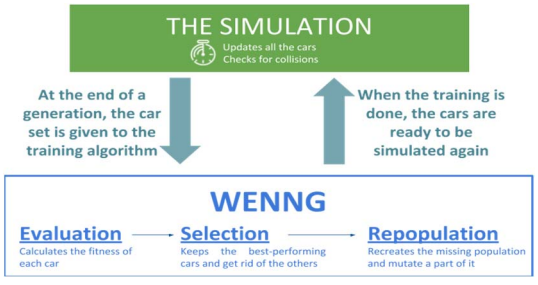

Figure 2. The three phases of the WENNG algorithm

The evaluation is the process of attributing a fitness score, as described in section 3.d, to each car being trained to determine those that are considered good for reproduction. The selection is the process of eliminating the worst performing cars and thus keeping the best ones. This process chooses cars to be eliminated using a weighted random distribution that we presented in [4]. The last phase of the algorithm randomly chooses two of the

remaining cars and combines them by randomly mixing the weights of their ANN. This last step is repeated until all the cars that were eliminated are replaced.

Note that this technique takes into account a lot of randomness. The weights of the initial networks and eliminated cars are chosen randomly. In addition, the repopulation takes a random parent and mixes the weights randomly.

V. Overview of LASP

Most deep learning algorithm simulation platforms like the ones in the related work section do not offer a way to easily share the developed algorithms. Scientists wanting to implement a deep learning algorithm with these platforms must reproduce and adapt the code and the infrastructure behind the algorithm to make it work with the environment. Thus, we added a way to share and download algorithms integrated on LAOP. The Learning Algorithm Sharing Platform (LASP) is a key part of the enhanced version of LAOP and provides a straightforward approach for solving this problem.

LASP is an online platform where scientists can share their deep learning algorithm built with LAOP. This online platform is directly integrated on LAOP, which allows researchers to easily and quickly download algorithms made by others. This makes comparison between algorithms easy. This is an important improvement over the first version of LAOP because we think comparing algorithms plays a key role in the constant research of better algorithms.

VI. Simulations Results

The main goal of the platform is to give researchers in the field of artificial intelligence the ability to compare their own algorithms with others. In this section, we demonstrate how this process works by comparing FUCONN with our newly proposed algorithm MHUB.

(a) Parameters

The parameters that were used for our experiment are as follows. First, each of the cars had five proximity sensors with a length of 300 pixels. The FUCONN algorithm ran with a car density of 100 vehicles and since the MHUB algorithm works with one car, controlled by the user, its car density was set to one.

(b) Comparison of the fitness of FUCONN and MHUB

This experiment compares FUCONN and MHUB while learning. To achieve this, we controlled the MHUB algorithm for 47 episodes. This translates to one hour of manually controlling the car in the maze. Like explained earlier, the length of an episode depends on the movement of the vehicles. If the vehicle loop backs on the path, it will get fewer points than a vehicle that tries new paths. After training MHUB, we trained the FUCONN algorithm for the same number of episodes. After each episode, we tested the best performing car in the past simulation on 100 randomly generated mazes to get the average score of the algorithm at that specific episode. We then tested them on multiple generated mazes because some mazes are easier to solve than others. If we only tested our algorithms on a small amount of mazes, the results would not be reliable, because it would depend a lot on the randomness of the generated mazes. Figure 3 shows an example of an easy and a hard maze to solve. In this figure, the one on the left is easier to solve since there is not a wall in front of the car.

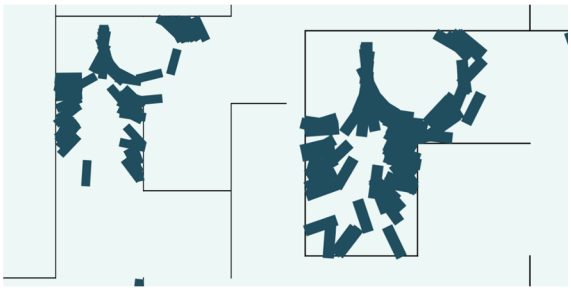

Figure 3. On the left is an example of an easy maze to solve. On the right, a hard one

The figure 4 shows the evolution of the score of each algorithm for the 47 episodes. We can see that the score of FUCONN increases quickly at the beginning but when it stabilises at around episode five, MHUB surpasses it.

B.1 Results

FUCONN gets a big jump of fitness and then stabilizes at episode five and twenty-six. This can be explained by the way this algorithm learns. At the end of each episode, the best performing cars are kept and a reproduction is taking place. As the reproduction process is based on chance, increasing the overall performance of the algorithm is based on luck. Between these events, performance is constant. This is why we only see jumps in performance and relatively constant values between these jumps.

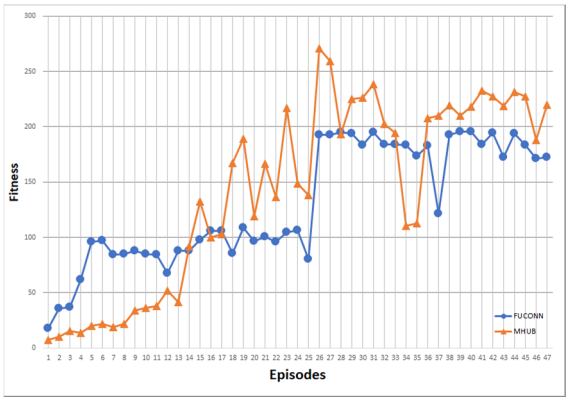

Figure 4. Score of both algorithms, FUCONN and MHUB during their learning process

B.2 Comparison of the performance of MHUB and FUCONN after being trained

The previous subsection compared MHUB and FUCONN during their training session. This analysis is to determine

which one was performing better at the end of our tests. We trained both algorithms until we did not see any improvements. This translates to 1h (47 episodes) of training with the MHUB algorithm and 120 episodes for the FUCONN algorithm. We then performed an evaluation with 100 randomly generated mazes. FUCONN got a score of 282 and MHUB got a score of 220.

With these results, we cannot conclude that reinforcement-learning techniques are better than supervised ones. Techniques in these two fields vary greatly and both have their advantages and flaws. Claiming one is better than the other would need more data and research.

Our tests concluded that the FUCONN algorithm surpassed the MHUB algorithm by 28.1 %. Multiple factors can influence these results. One of them is the reliability of the data collected by MHUB. When a human is driving the car, the driver can make errors that MHUB will consider as good actions. Another one is the criteria for stopping the training and considering the algorithms as fully trained. In our case, this was purely subjective. We stopped the training when we did not see any improvements in the cars being simulated.

B.3. Results

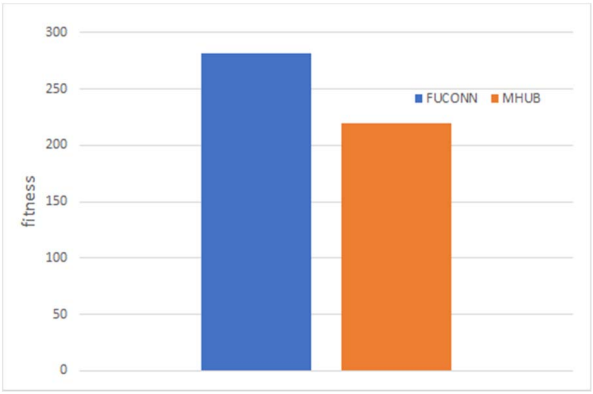

Figure 5. Comparison between the average score MHUB and FUCONN got

VII. Conclusion and future works

In this paper, we presented techniques to compare algorithms for self-driving cars. We proposed an implementation of this technique called LAOP. We also showcased the improvements made to LAOP. With this implementation, we compared two algorithms, FUCONN and MHUB and showed that FUCONN was 28.1% more performant than MHUB. We also compared both algorithms during their learning phase and found that FUCCON has a faster start than MHUB but quickly gets surpassed.

We continually improve LAOP and we hope that the enhancements we highlighted in this paper will attract more people on our platform. For future work, it would be interesting to create new environments tackling completely different problems and then compare our algorithms to see how generalized our algorithms are. It would also be interesting to improve LAOP 2.0 by allowing the algorithms to run on multiple threads and/or GPU to increase simulation speed, thus being able to compare bigger timeframes.

References

[1] Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, et al. “End to End Learning for Self-Driving Cars.” ArXiv:1604.07316 [Cs], April 25, 2016. http://arxiv.org/abs/1604.07316.

[2]Yang, Zhengyuan, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. “End-to-End Multi-Modal Multi-Task Vehicle Control for Self-Driving Cars with Visual Perception.” *ArXiv:1801.06734 [Cs]*, February 2, 2018. http://arxiv.org/abs/1801.06734.

[3] De S., Varuna, Jamie R. and Ahmet K.. “Robust Fusion of LiDAR and Wide-Angle Camera Data for Autonomous Mobile Robots.” ArXiv:1710.06230 [Cs], August 23, 2018. http://arxiv.org/abs/1710.06230.

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

[5] LAOP by L. Oest O’Leary, C. Bisaillon and J. Rezgui on GitHub, https://github.com/lool01/LAOP, [last visit and update 01/05/2020].

[6] J. Rezgui, L. Oest O’Leary, C. Bisaillon and L. Chaari, "Training Genetic Neural Networks Algorithms for Autonomous Cars with the LAOP Platform", IEEE IWCMC 2019 Tangier, Morocco.

[7] Open AI. (2019, April 14). http://www.openai.com

[8] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, “Open AI gym”, arXiv:1606.01540

[9] Amazon Web Services, Inc. “Amazon SageMaker.” Accessed January 25, 2020. https://aws.amazon.com/sagemaker/

[10] Shraddha M., Bhargav P., Nikhil T. and Kelly B. “Simulation of Self Driving Car”, Available online: https://bhargav265.github.io/Simulation_of_self_driving_car/ArtificialLifeReport.pdf

[11] Multiple contributors. “deeplearning4j”, GitHub repository https://github.com/deeplearning4j/deeplearning4j [last visit and update 01/05/2020].