An End-to-End Curriculum Learning Approach for Autonomous Driving Scenarios

Luca Anzalone, Paola Barra, Silvio Barra, Aniello Castiglione, *Member, IEEE*, and Michele Nappi, *Senior Member, IEEE*

***Abstract*— In this work, we combine *Curriculum Learning* with *Deep Reinforcement Learning* to learn without any prior domain knowledge, an end-to-end competitive driving policy for the CARLA autonomous driving simulator. To our knowledge, we are the first to provide consistent results of our driving policy on *all* towns available in CARLA. Our approach divides the reinforcement learning phase into multiple *stages* of increasing difficulty, such that our agent is guided towards learning an increasingly better driving policy. The agent architecture comprises various neural networks that complements the main convolutional backbone, represented by a *ShuffleNet V2*. Further contributions are given by (i) the proposal of a novel value decomposition scheme for learning the value function in a stable way and (ii) an ad-hoc function for normalizing the growth in size of the gradients. We show both quantitative and qualitative results of the learned driving policy.**

***Index Terms*— Autonomous driving, CARLA simulator, automotive, deep reinforcement learning, curriculum learning.**

I. Introduction

AUTONOMOUS Driving (AD) technology promises to change the way we travel. Thanks to the emerging automotive applications, Autonomous Vehicles (AV) will be able to recognize the road and the driving context so to plan the route by monitoring the dynamics of the other vehicles and subjects within the scene. Thanks to AV, people will be able to travel from place to place in a safer, more environmentally friendly and even more time-efficient way. These new technologies are expected to reduce road fatalities, pollution and have greater autonomy.

Such complex goals can be only achieved by highly autonomous vehicles, classified as level 4 (high automation) and level 5 (full automation) by the SAE J3016 standard [1]. High-to-full autonomous vehicles must master tasks known as *perception*, *planning*, and *control* [2], [3]. *Perception* refers to the ability of an autonomous system to collect information and extract relevant knowledge from the environment. In order to do so, the autonomous vehicles need to understand the driving scenario (*environmental perception*), to compute its pose and motion (*localization*), and to determine which portions of the driving space are occupied by other objects (*occupancy grids*). *Planning* relies on the output of the perception component to devise and obstacle-free route, that the vehicle has to follow to avoid any collision while reaching its indented destination. The planned route is made of high-level commands that do not tell the vehicle's software how to actually implement them in terms of torques and forces. Finally, *motion control* does account for this, converting high-levels commands to low-levels actions, consisting of specific torques and forces values to be applied to the vehicle's actuators in order to make it move and steer properly.

For such purpose, both level 4 and 5 autonomous vehicles are equipped with a variety of *exteroceptive sensors*, like cameras, LiDAR, RADAR, and ultrasonic sensors, to perceive the external environment including dynamic and static objects, and *proprioceptive sensors*, like IMUs, tachometers and *altimeters*, for internal vehicle state monitoring [4]. Moreover, high sensor redundancy along with *sensor fusion* are often necessary to achieve improved performance and high robustness especially in degraded driving and weather conditions.

The tasks of perception, planning and control can be solved in isolation or jointly. In the isolated approach it is interpreted with a *modular pipeline* in which each module is separate and performs a specific task [4]. The resulting system suffers from error propagation: the modules are designed by humans and therefore potentially imperfect; every small error would propagate in the system joining with any errors in other modules. Basically, the isolated approach is not optimal and not reliable. These weaknesses motivate the choice of the *end-to-end driving* paradigm. With end-to-end guidance, the perception, planning and control tasks are solved jointly and are not presented explicitly. These systems have a more functional design and are easier to develop and maintain.

In general, we can distinguish various categories of system architectures for autonomous vehicle design which also accounts (or not) for connectivity among vehicles [4]:

- **Ego-only systems** (or *standalone* vehicles) do not share information among other autonomous vehicles.

Manuscript received 30 June 2021; revised 27 October 2021; accepted 22 February 2022. Date of publication 26 May 2022; date of current version 11 October 2022. This work was supported in part by PRIN 2017 PREVUE: "PRediction of activities and Events by Vision in an Urban Environment," through the Italian Ministry of Education, University and Research, under Grant 2017N2RK7K. The Associate Editor for this article was B. B. Gupta. *(Corresponding author: Aniello Castiglione.)*

Luca Anzalone is with the Department of Physics and Astronomy (DIFA), University of Bologna, 40127 Bologna, Italy (e-mail: luca.anzalone2@unibo.it).

Paola Barra is with the Department of Computer Science, Sapienza University of Rome, 00185 Rome, Italy (e-mail: barra@di.uniroma1.it).

Silvio Barra is with the Department of Electrical and Information Technology Engineering (DIETI), University of Naples "Federico II", 80138 Naples, Italy (e-mail: silvio.barra@unina.it).

Aniello Castiglione is with the Department of Science and Technology (DIST), University of Naples "Parthenope", 80133 Naples, Italy (e-mail: castiglione@ieee.org).

Michele Nappi is with the Department of Computer Science, University of Salerno, 84084 Salerno, Italy (e-mail: mnappi@unisa.it).

Digital Object Identifier 10.1109/TITS.2022.3160673

A standalone vehicle uses only its knowledge to devise driving decisions. The lack of connectivity, makes this category of AVs simpler to design compared to vehicles that are connected together.

- **Connected systems** are able to distribute the basic operations of automated driving among other autonomous vehicles, thus forming a *connected multi-agent system*. In this way, vehicles can share detailed driving information and use such additional information to perform better decisions. Communication among vehicles requires a specific infrastructure and communication protocols, other than being able to efficiently transmit and store large amounts of data.
- **Modular systems** are structured as a pipeline of separate components (as discussed previously), each of them solving a specific task. The main advantage is that the complex problem of autonomous driving can be decomposed in smaller and easier-to-solve set of problems.
- **End-to-end driving** generate ego-motion directly from (raw) sensory inputs (e.g. RGB camera images), without the need to design any intermediate module. Ego-motion can be either the continuous operation of steering wheel and pedals (i.e. acceleration and breaking) or a discrete set of actions. End-to-end driving is simple to implement, but often leads to less interpretable systems.

Imitation Learning [5], [6] is the preferred approach for end-to-end driving, given its design simplicity and optimization stability, despite requiring a considerable amount of *expert data* for learning a competitive policy. Deep Reinforcement Learning (RL) is gaining interest for its encouraging results in the field [7], [8], without requiring to collect expert trajectories: just a real or simulated environment (e.g. CARLA [9], or AirSim [10]) is needed, instead. Moreover, RL can potentially discover better-than-expert behavior since it maximizes the agent's performance with respect a designed reward function.

In this paper, we provide the following contributions:

- We combine the *Proximal Policy Optimization* (PPO) [11] algorithm with Curriculum Learning [12], showing how to learn an end-to-end urban driving policy for the CARLA driving simulator [9].
- We evaluate our curriculum-based agent on various metrics, towns, weather conditions, and traffic scenarios. To our knowledge, we are the first to demonstrate consistent results on *all* towns provided by CARLA, by just training the agent on only *one* town.
- Moreover, we point out two important sources of instability in reinforcement learning algorithms: learning the value function $V(s)$, and normalizing the estimated advantage function $A(s, a)$.
- Finally, we provide two novel techniques to solve these issues. The two methods can be applied to any value-based RL algorithm, as well as actor-critic algorithms. More notably, the same technique we use to learn the value function is general enough to be employed in almost any ML regression problem.

The paper is organized as follows: Section II defines and describes the related works in the topic, categorizing them in (i) Autonomous Driving approaches based on Deep Learning techniques, (ii) Reinforcement Learning for Autonomous Driving and (iii) Autonomous Driving Simulators. In Section III several formalisms and definitions are proposed, so to allow the reader to ease the understanding of the background context of the paper. In Section IV the proposed approach is presented. Sections V shows the obtained results on the CARLA *Towns*. Finally, Section VI concludes the paper.

II. Related Work

A. Deep Learning-Based Autonomous Driving

Deep learning-based end-to-end driving systems aim to achieve human-like driving simply by learning a mapping function from inputs to output targets, so being able to *imitate* human experts. These inputs are often (monocular) camera images, while the targets can be quantities like the steering angle, the vehicle's speed, the route-following direction, throttle and breaking values, or even high-level commands.

Reference [13] trained a convolutional neural network to map raw pixels from a single front-facing camera directly to steering commands. The authors managed to drive in traffic on local roads, on highways, and even in areas with unclear visual guidance. To correct the vehicle drifting from the ground-truth trajectory, the authors employed two additional cameras to record left and right shifts. The authors evaluated their system by measuring the *autonomy metric*, being autonomous 98% of the time. To mitigate this shifting problem, [14] developed a sensor setup that provides a 360-degree view of the area surrounding the vehicle by using eight cameras. Their driving model uses multiple Convolutional Neural Networks (CNNs) as feature encoders, four Long-short Term Memory (LSTM) recurrent networks [15] as temporal encoders, and a fully-connected network to incorporate map information. Their system is trained to minimize the mean squared error (MSE) against speed and steering angle.

Reference [5] propose to *condition* the imitation learning procedure on a high-level routing command (i.e. a one-hot encoded vector), such that trained policies can be controlled at test time by a passenger or by a topological planner. The authors evaluated the approach in a simulated urban environment provided by the CARLA driving simulator [9] and on a physical system: a 1/5-scale truck. For goal-based navigation they recorded a success rate of 88% in `Town 1` (training scenario), and of 64% in `Town 2` (testing scenario); two of the simplest towns available.

End-to-end behavioral cloning is appealing for its simplicity and scalability but there are limitations [6], such as: *dataset bias* and *overfitting* when data is not diverse enough, *generalization* issues towards dynamic objects seen during training, and *domain shifting* between the off-line training experience and the on-line behavior. Despite these limitations, behavioral cloning can still achieve state-of-the-art results as demonstrated by [6]. In fact, the authors proposed a ResNet-based [16] architecture with a speed prediction branch. According to them, in presence of large amounts of data a deep model can reduce both bias and variance over data, also having better generalization performances on

learning reactions to dynamic objects and traffic lights in complex urban environments. The authors also proposed a novel CARLA driving benchmark, called *NoCrash*, in which the ability of the ego vehicle is tested on three urban scenarios with different weather conditions: *empty town* with no dynamic objects, *regular traffic* with a moderate amount of cars and pedestrians, and *dense traffic* with a large number of vehicles and pedestrians.

Reference [17] proposed the first *direct perception* method - an emerging paradigm that combines both end-to-end learning and control algorithms - named Conditional Affordance Learning (CAL), to handle traffic lights and speed signs by using image-level labels, as well as smooth car-following, resulting in a significant reduction of traffic accidents in simulation. Their CAL agent consists of a neural network that predicts six types of affordances from input observation, and a lateral and longitudinal controller which predicts the throttle, brake, and steering values.

Reference [18] proposed the first *interpretable* neural motion planner for learning to drive autonomously in complex urban scenarios that include traffic-light handling, yielding, and interactions with multiple road-users. Their model employs a convolutional backbone to predict the bounding boxes of other actors, as well as a *space-time cost volume* for planning. The input representation consists of Lidar point clouds coupled with annotated HD maps of the road. The space-time cost volume represents the goodness of each location that the self-driving car can take within a planning horizon. Their model is trained end-to-end with a multi-task objective: the *planning loss* encourages the minimum cost plan to be similar to the trajectory performed by human demonstrators, and the *perception loss* encourages the intermediate representations to produce accurate 3D detection and motion forecasting. According to the authors, the combination of these two losses ensures the interpretability of the intermediate representations.

B. Reinforcement Learning-Based Autonomous Driving

Reference [8] demonstrated the first application of deep reinforcement learning to autonomous driving. Their model is able to learn a policy for lane following in a handful of training episodes using a single monocular image as input. The authors used Deep Deterministic Policy Gradient (DDPG) algorithm [19] with prioritized experience replay [20] with all exploration and optimization performed on-vehicle. Their state space consists of monocular camera images compressed by a learned Variational Auto-Encoder (VAE) [21] together with the observed vehicle speed and steering angle. The authors defined a two-dimensional continuous action space: steering angle, and speed set-point. The authors utilize a 250 meter section of road for real-world driving experiments. Their best performing model is capable of solving a simple lane following driving task in *half an hour*.

Reference [7] proposes Controllable Imitative Reinforcement Learning (CIRL) to learn a diving policy based only on vision inputs from the CARLA simulator [9]. CIRL adopts a two-stage learning procedure: a first imitation stage pretrains the actor's network on ground-truth actions recorded from human driving videos, and the subsequent reinforcement learning stage employs DDPG [19] to improve the driving policy. According to the authors, the first imitation stage is necessary to prevent DDPG to fall in local optima due to poor exploration. CIRL uses a four-branch network with a speed prediction branch, similar to [6]. The authors conducted experiments on the CARLA simulator benchmark, showing that the CIRL performance are comparable to the best imitation learning methods, such as CIL [5], CAL [17], and CIRLS [6].

Often, training a competitive driving policy from high-dimensional observations is often too difficult or expensive for RL. [22] propose to *visual encode* the perception and routing information the agent receives into a *bird-view image*, which is further compressed by a VAE [21]. To reduce training complexity the authors employed the *frame-skip trick*, in which each action made by the ego-vehicle is repeated for subsequent $k = 4$ frames. The authors evaluated their approach on CARLA [9], specifically on a challenging roundabout scenario in `Town 3`. They compared three RL algorithms: Double DQN [23], TD3 [24], and SAC [25]. The latter achieved the best performance.

Reference [26] proposed a *multi-objective* DQN agent motivated by the fact that a multi-objective approach can help overcome the difficulties of designing a scalar reward that properly weighs each performance criteria. Furthermore, the authors suggest that when each aspect is learned separately, it is possible to choose which aspect to explore in a given state. In particular, they learned a separate agent for each objective which, collectively, form a *combined policy* that takes all these objectives into account. The authors trained the agent on two four-way intersecting roads with random surrounding traffic provided by the SUMO traffic simulator [27], demonstrating very low infraction rate.

C. Autonomous Driving Simulators

Autonomous driving research requires a considerable amount of diversified data, collected on a variety of driving scenarios with different weather conditions as well. Collecting such amount of data in the real-world is difficult, time-consuming, and costly. Moreover, driving datasets often focus only on specific aspects of the driving task, also collected with specific sensor modalities (e.g. RGB cameras vs Lidar sensors).

An increasing popular alternative to real-world data are *autonomous driving simulators*. Modern driving simulators like CARLA (Car Learning to Act) [9], and AirSim [10] provide realistic 3D graphics and physics simulation, traffic management, weather conditions, a variety of sensors, pedestrian management, different vehicles, and various driving scenarios as well. In particular, AirSim also supports autonomous aerial vehicles, like drones. These kind of simulators are very flexible providing an easy way to collect data in different driving scenarios, weather conditions, with different vehicles and sensor modalities. TORCS (The Open Racing Car Simulator) [28] is a modular, multi-agent car simulator that focus on racing scenarios, instead. Compared to CARLA and

AirSim, TORCS has a lower-quality graphics, no traffic and pedestrian simulation, and a limited set of sensors. Other kind of driving simulators focus solely on traffic simulation. SUMO (Simulation of Urban Mobility) [27] is a microscopic traffic simulation tool that models each vehicle and their dynamics individually. In particular, SUMO can even simulate railways and the CO_2 emissions of individual vehicles.

III. Background

In this section we provide basic formalism and results about Reinforcement Learning [29], Generalized Advantage Estimation [30], and Proximal Policy Optimization [11], needed for understanding and developing the subsequent sections.

A. Reinforcement Learning

Reinforcement Learning (RL) [29] is a learning paradigm to tackle decision-making problems that provides a formalism for modeling behavior, in which a software or physical *agent* learns how to take optimal *actions* within an *environment* (i.e. a real or simulated world) by trial and error, guided only by positives or negatives scalar *reward* signals (sometimes called *reinforcements*).

Formally, an environment is a *Markov Decision Process* (MDP) represented by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, in which: $\mathcal{S}$ is the *state space*, $\mathcal{A}$ is the *action space*, $\mathcal{P}(s' \mid s, a)$ is the *transition model* (also called the *environment dynamics*) with which is possible to predict the evolution of the environment's state, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward function*, finally $\gamma \in (0, 1]$ is the *discount factor*.

The state space defines all the possible states $s \in \mathcal{S}$ (of the environment) that can be experienced by the agent. Instead, the action space depicts all the possible actions $a \in \mathcal{A}$ that the agent can predict. If the state space is not *fully-observable*, the agent perceives observations $o \in \mathcal{O}$, instead, which are yield by the environment itself. The *observation space* $\mathcal{O}$ contains only a partial amount of information described by $\mathcal{S}$, the others (such as the environment's internal stat) are hidden. In order to recover such hidden information the agent usually retains (or processes somehow) the full (or partial) *history* of the previous observations, i.e. $o_{1:t}$, until the current timestep t. This setting is usually referred to be a *partially-observable Markov decision process* (POMDP).

The agent derives actions according to its *policy* $\pi : \mathcal{S} \to \mathcal{A}$ which can be either deterministic $a_t = \pi(s_t)$ or stochastic $\pi(a_t|s_t)$ mapping states s_t to actions a_t. Note that in a partially-observable setting (i.e. POMDP) the true states are not available to the agent, which derives actions by conditioning on (one or more) past observations instead: $\pi(a_t \mid o_{j:t})$, where the index j ($j \leq t$) indicates how much past observations are considered. For our purposes, we restrict the policy to be a *Deep Neural Network* (DNN) [31], π_θ with learnable parameters θ, that samples actions from a probability distribution, i.e. $a_t \sim \pi_\theta(\cdot|s_t)$. In our case, our agent predicts two continuous actions so we need to sample them from a *continuous* probability distribution like a Gaussian. Motivated by [32], we use a Beta distribution instead, which, apart from outperforming the Gaussian distribution, it is particularly suited for continuous actions that are also bounded.

In order to learn the desired behavior, the agent has to interact with the target environment: at the first timestep ($t = 0$) the environment provides the agent with an *initial state* $s_0 \sim \rho(s_0)$ sampled from the *initial state distribution* $\rho(s_0)$, usually implicitly defined by the environment. Then, the agent uses its policy to predict and execute the actions a_0 affecting the environment, resulting in state s_1 according to the environment dynamics, i.e. $s_1 \sim \mathcal{P}(\cdot \mid s_0, a_0)$. Consequently, the environment evaluates the newly reached state s_1 with its reward function also providing the agent the respective *immediate reward* $r_0 = r(s_0, a_0)$. Then, the interaction loop repeats for the next timestep until either a final state or the maximum number of timesteps have been reached. In general, the interaction loop proceeds as follows: at a generic timestep t the agent experiences a state s_t, then it computes actions a_t resulting in state s_{t+1} for which it receives a reward $r_t = r(s_t, a_t)$ from the environment. In practice, we consider *finite horizon* episodes of maximum length T.

B. Proximal Policy Optimization

Proximal Policy Optimization (PPO) [11] is a model-free RL algorithm from the policy optimization family that aims to learn policies in a faster, more efficient, and more robust way compared to vanilla policy gradient [33], and TRPO [34]. In general, the aim of RL algorithms is to *indirectly* maximize the *performance objective* $J(\theta)$ in order to maximize the agent's performance on the given task:

$$J(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T-1} \gamma^t r_t\Big], \tag{1}$$

Maximizing the performance objective $J(\theta)$ means maximizing the *expected* sum of discounted rewards, seeking for a policy $\pi^\star = \arg\max_\pi J(\theta)$ that achieves maximal performance (i.e. $\sum_t r_t$ is maximal). The objective (1) is stochastic (since the rewards results from sampled states and actions by following π), apart from being not directly differentiable. Hence, policy optimization algorithms (likewise other RL methods) optimize a *surrogate objective* $\tilde{J}(\theta)$, instead, called the *policy gradient*:

$$\nabla_\theta \tilde{J}(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A(s_t, a_t)\Big], \tag{2}$$

where π_θ is a policy parameterized by θ, and $A(s, a)$ is the advantage function. The PPO algorithm optimizes a slightly different policy gradient objective to maximize $J(\theta)$. In particular we utilize the following *clipping objective* variant (borrowing notation from [11]):

$$\mathcal{L}_{\text{clip}}(\theta) = \mathbb{E}_t\Big[\min\Big(\text{ratio}_t(\theta)\hat{A}_t,\ \text{clip}(\text{ratio}_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\Big] \tag{3}$$

where: $\text{ratio}_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ denotes the probability ratio between the current π_θ and old policy $\pi_{\theta_{\text{old}}}$, $\hat{A}_t$ represents the advantages estimated by using GAE, lastly the function

clip(·) truncates $\text{ratio}_t(\theta)$ at $1-\epsilon$ if the advantages $\hat{A}_t$ are negatives, otherwise the ratio is clipped at $1+\epsilon$. Lastly, the hyper-parameter ϵ is usually set to 0.2. In practice, we also add an *entropy regularization* term $\mathcal{H}[\pi_\theta]$ to the objective (3) to ensure diverse enough actions.

Equation (3) depicts the clipping objective used by PPO to improve the policy's parameters θ, moreover the clipping function ensures the current policy to be not too different from the old policy so that divergent behavior is less likely. Finally, the agent's parameters θ got updated by performing multiple gradient descent steps (usually by using the *Adam optimizer* [35]) with respect to (3). The update rule looks like the following:

$$\theta' \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{clip}}(\theta)$$

where θ' are the new parameters, and η is the learning rate.

C. Generalized Advantage Estimation

Many RL algorithms belonging to the policy optimization family – REINFORCE [33], TRPO [34], PPO [11], and A3C [36] – require to estimate the *advantage function* $A(s,a)$ in order to learn the desired behavior. The advantage function $A(s,a) = Q(s,a) - V(s)$ is defined as being the difference between the *action-value function* $Q(s,a)$ and the *state-value function* $V(s)$:

$$Q(s,a) = r(s,a) + \gamma \cdot \mathbb{E}_{s' \sim \mathcal{P}(s,a)}\big[V(s')\big] \tag{4}$$

$$V(s) = \mathbb{E}_{a \sim \pi(s)}\big[Q(s,a)\big] \tag{5}$$

Intuitively, the advantage function tells us how much the action a is better or worse than the average action while being in state s. In particular, the action a is better-than-average if $Q(s,a) > V(s)$, and is worse-than-average if $Q(s,a) < V(s)$.

To estimate the advantage function is only necessary to learn *either* the state-value function $V(s)$ or the action-value function $Q(s,a)$, since both functions can be defined in terms of the other. In particular we use the *Generalized Advantage Estimation* (GAE) [30] technique, which is an exponentially-weighted estimator of the advantage function that only requires a learned state-value function. The GAE estimator has two hyper-parameters $\gamma \in (0,1]$ and $\lambda \in [0,1]$ which allows us to trade variance for bias. Finally, the generalized advantage estimator $\text{GAE}(\gamma, \lambda)$ is defined as follows:

$$\begin{aligned} A_t^{\text{GAE}(\gamma,\lambda)} &= (1-\lambda)\Big(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \cdots\Big) \\ &= \sum_{k=0}^{T-1} (\gamma\lambda)^k \delta_{t+k} \end{aligned}$$

it builds from the n-step return estimator $A_t^{(n)}$ which is defined as the sum of n TD-residual δ_{t+k} terms:

$$\begin{aligned} A_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k \delta_{t+k} = -V_\phi(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \\ + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V_\phi(s_{t+n}) \end{aligned}$$

where V_ϕ is a learned state-value function.

IV. Method

A. Learning Environment

The learning environment $\mathcal{E} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, formally a Partially Observable Markov Decision Process (POMDP), defines the task that agent has to complete. This environment $\mathcal{E}$ was built by combining the CARLA driving simulator (version 0.9.9) [9] and the OpenAI's gym library [37]:

- **State space** $\mathcal{S}$ is implicitly defined by CARLA, containing ground-truth information about the whole world. The agent cannot observe the environment's state $s_t \in \mathcal{S}$, thus the states are *hidden*. At each timestep t, the state s_t yields the corresponding *observation* o_t, which is what the agent observes.
- **Observation space** $\mathcal{O}$: similarly to [38], an observation $o_t \in \mathcal{O}$ is a stack of $K = 4$ sets of tensors from the last K timesteps. Specifically, $o_t = \big\{[I, G, V, N]_k\big\}_{k=1}^{4}$, where: I is a $90 \times 360 \times 3$ image obtained by concatenating (along the width axis) three $90 \times 120 \times 3$ images from left, middle, and right RGB camera sensors, G is a 9-dimensional vector that encodes *road features*, V is a 4-dimensional vector that embeds *vehicle features*, and, lastly, N is a 5-dimensional vector that contains *navigational features*. The road features G comprise: three Boolean values (`is_intersection`, `is_junction`, and `is_at_traffic_light`), the speed limit (a float), and the traffic light state (a 5-dimensional one-hot vector). The vehicle features V contains: the current vehicle speed, the actual throttle and brake values, and the vehicle similarity score $v_{\text{sim}} \in [-1, 1]$ w.r.t. the next planned route waypoint (center of a road segment). Lastly, the navigational features N include $n = 5$ distances between the actual location of the vehicle and the next n planned route waypoints.
- **Action space** $\mathcal{A}$:composed of two continuous actions with value in the range $[-1, +1]$. These two actions are the *accelerator or brake* value, and the *steering angle*.
- **Transition dynamics** $\mathcal{P}(s_{t+1} \mid s_t, a_t)$: defines how the environment's state $s_t \in \mathcal{S}$ evolves in time due to the application of the predicted actions $a_t \in \mathcal{A}$. The transition dynamics is defined by CARLA, and is not explicitly learned by the agent (model-free setting).
- **Reward function** r: penalizes any collision, as well as following the wrong planned route. Compared to other methods [7], [22] our reward function is simple, as it relates vehicle's speed, direction, infractions, and collisions in an intuitive way, thus avoiding the need to optimally weight many different terms:

$$r_t = \begin{cases} -c_p & \text{if collision,} \\ s_{\text{limit}} - v_{\text{speed}} & \text{if } v_{\text{speed}} > s_{\text{limit}}, \\ \dfrac{v_{\text{speed}} \cdot v_{\text{sim}}}{(d_w/2)^2} & \text{otherwise} \end{cases} \tag{6}$$

where v_{speed} is the vehicle's speed, v_{sim} is the vehicle's (cosine) similarity with next waypoint w, d_w is the l_2 distance between the vehicle's position and w, s_{limit} is the speed limit, lastly c_p is the penalty for colliding with objects, vehicles, and pedestrians.

Fig. 1. Example results of applying data augmentation. The original image is at the top-left corner.

- **Discount factor** $\gamma \in (0, 1]$: future rewards are discounted by a factor of γ at each timestep.

B. Data Augmentation

As demonstrated by previous work [5], [17], [39] data augmentation is crucial to let the agent generalize across different towns and weather conditions. Similarly to [5], the augmentations used are: *color distortion* (i.e. changes in contrast, brightness, saturation, and hue), *Gaussian blur*, *Gaussian noise*, *salt-and-pepper noise*, *cutout*, and *coarse dropout*. Each augmentation function is applied with a certain probability and intensity (see Fig. 1).

Geometrical transformations, commonly used for image detection tasks, including horizontal or vertical flipping, rotation, and shearing, are not applied in this case since they would significantly alter the driving scene.

Note that data augmentation have been only used in the last two stages of the reinforced curriculum learning procedure (more details in section IV-F).

C. Agent Architecture

The agent is implemented by a deep neural network [31] that takes the current observation o_t as input, and outputs the next action $a_t \sim \pi_\theta(z_t)$ along its value $v_t = V_\phi(z_t)$, where $z_t = P_\psi(o_t)$. The deep neural network represented by the agent has two branches: the *policy branch* π_θ with parameters θ (the actor), and the *value branch* V_ϕ with parameters ϕ (the critic). The policy branch samples a actions from a Beta distribution as motivated by [32]. The value branch outputs the value v of the states s that are used to estimate the advantage function $A(s, a)$ with the GAE [30] technique. Both branches share a common neural network P_ψ with parameters ψ, that processes observations o into an intermediate representation z. Since each observation o_t is a stack of 4 sets of tensors (see section IV-A), i.e. $o_t = [o_t^1, \dots, o_t^4]$, the network P_ψ is applied sequentially on each o_t^i, yielding four z_t^i which are aggregated by Gated Recurrent Units (GRUs) [40] to obtain z_t. Moreover, P_ψ embeds a ShuffleNet v2 [41] to process image data. Finally, both V_ϕ and π_θ are feed-forward NNs with two layers with 320 units SiLU-activated [42] and batch normalization [43].

The overall architecture of the agent is depicted in Fig. 2. The blue rectangles indicate fully-connected (or dense) layers. The blue circle, i.e. $\oplus$, denotes layer concatenation along the first dimension (or axis), where the batch dimension is

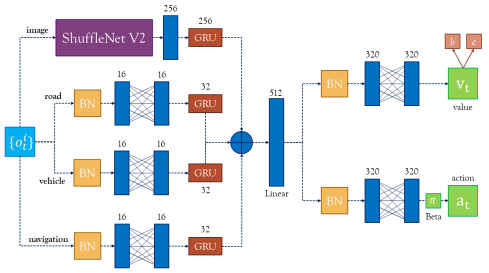


Fig. 2. The neural network architecture of the proposed agent (with minor omissions). The first half depicts the shared network P_ψ, while the second half shows, respectively from top to bottom, the value V_ψ and policy π_θ branches. At the center, the outputs of the first half of the network is first concatenated and then linearly combined, before feeding it to both the value and policy branches.

at axis zero. The shared network P_ψ (first half) processes each component of the observation tensor o_t^i separately, which are independently aggregated by GRU layers [40] into single vectors. Then, the output of the concatenation is linearly combined and fed to the two branches. Lastly, values are decomposes into two numbers, bases b and exponents e, as motivated in the following section.

D. Learning the Value Function

The value function is learned by minimizing the squared loss $\mathcal{L}_v(\phi) = \|v - R\|_2^2$ of the network estimate of the values $v = [v_t]_{t=0}^{T-1}$, towards the true returns $R = [R_t]_{t=0}^{T-1}$, where each return $R_t = \sum_{i=t}^{T-1} \gamma^i r_i$ is the discounted sum of rewards from timestep t to the end of the episode $T - 1$.

Let's notice that when the quantity $\|v - R\|_2^2$ is large, because the estimate v is far from the ground-truth R, also (the norm of) its gradient $\nabla_\phi \mathcal{L}_v(\phi)$ is large, and so the parameters ϕ got a big update that can cause training to be less stable. Both values and returns are normalized to have zero mean and unitary variance, this is a commonly used practice to reduce variance, so that the magnitude of the error is always small. It is not known in advance to what proportion the values and returns are normalized to avoid bias, for this reason this approach is biased: *the scale of such quantities changes as the performance of the agent improves*.

The following outlines the approach used to learn the function solidly and accurately, even without any normalization bias: both values v and returns R are respectively divided into bases $b_v, b_R \in [-1, 1]$ and exponents $e_v, e_R \in [0, k]$ such that

$$v = b_v \cdot 10^{e_v}$$

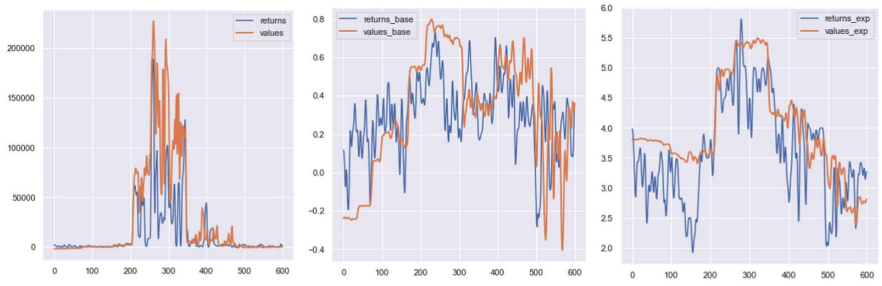


Fig. 3. Example of value function learned through base-exponent decomposition. In the leftmost plot, the learned value function compared to returns; in the center plot, the regression of bases; in the rightmost plot, the regression of exponents.

$$R = b_R \cdot 10^{e_R},$$

where $k \in \mathrm{N}$ is a positive constant that should be large enough to represent even the largest returns. For example, we set $k = 6$ so that even returns up to $\pm 10^6$ can be properly depicted. With such base-exponent decomposition, learning the value function is a matter of regressing both bases and exponents; the new loss function $\mathcal{L}'_v(\phi)$ is defined as follows:

$$\mathcal{L}'_v(\phi) = \sum_{t=0}^{T-1} \frac{(b_{v_t} - b_{R_t})^2}{4} + \frac{(e_{v_t} - e_{R_t})^2}{k^2} \tag{7}$$

Hence, even large errors now lie in a small interval because both the base b and exponent e take values in a small interval, and so the gradient $\nabla_\phi \mathcal{L}'_v(\phi)$ is always reasonably small, resulting in more stable training. Note that the bases b have a different scale from the exponents e, so we normalize them (by respectively dividing by 4 and k^2) such that they equally contribute to the loss value, once again avoiding the need to weight these two error terms. The normalizing coefficients are obtained by considering the worst case in the squared differences. Since the bases $b \in [-1, 1]$, the worst case (i.e. the larger error value) is given by $(1 - (-1))^2 = 4$: supposing $b_{v_t} = 1$ and $b_{R_t} = -1$ (or vice-versa). Similarly for the exponents $(0 - k)^2 = k^2$, since $e \in [0, k]$, again supposing $e_{v_t} = 0$ and $e_{R_t} = k$ (or vice-versa).

E. Sign-Preserving Advantage Normalization

The estimated advantages $\hat{A}_t$ directly affect the norm of the gradient $\nabla_\theta \mathcal{L}_{\text{clip}}(\theta)$ of the PPO's policy objective (3) as being a multiplicative factor. Consequently, if the advantages are large also the norm of $\nabla_\theta \mathcal{L}_{\text{clip}}(\theta)$ is large, resulting in a considerable change of the policy's parameters: resulting in a probable change of the agent behavior, which may easily diverge; otherwise we could lower the learning rate by several factors, potentially slowing-down training. Note that the magnitude of the advantages strictly depends on the quality of the learned value function, thus: *poorly estimated values imply large advantages*, since $\hat{A} \approx V_\phi(s) - R$, where V_ϕ is a learned value function and R the true returns. So, it is important to scale the advantages in a reasonable range without introducing any bias to stabilize learning (Fig. 4).

For such purpose we propose the *sign-preserving normalization* function which separately normalizes positive values from negative ones. The function is defined by the following TensorFlow 2 [44] code:

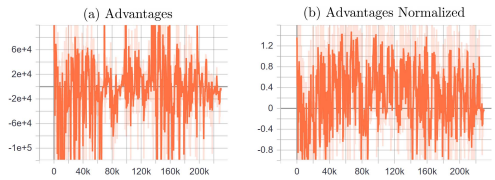


Fig. 4. Normalized advantages (b) now have a small scale, roughly in $[-1, 1]$. The magnitude of the original advantages (a) was much larger, in the order of 10^5. This ensures the policy gradient's norm to be small as well. Notice the scale of the normalized advantages is almost 10^4 times smaller. Moreover, our normalization scheme ensures the preservation of the sign, that is if in (a) some advantages were positive, they will be still positive after our normalization in (b).

```
def sign_preserving_norm (adv, eps=1e-3):
   adv_max = tf.reduce_max(adv)
   adv_min = tf.reduce_min(adv)

   # first, filter positives and negatives
   pos = adv * tf.cast(adv > 0, tf.float32)
   neg = adv * tf.cast(adv < 0, tf.float32)

   # then, normalize them separately
return (pos / (adv_max + eps)) +
       (neg / -(adv_min - eps))
```

Advantages normalized with the above function have the benefit to have the *same sign* (and, thus, meaning) of the original advantages (Fig. 5), while having a small and *controllable* scale which we argue contribute to stabilize training. Preserving the sign is an important property which avoids detrimental *gradient flipping* issues that cause ambiguity in the policy between better-than-average actions against worse-than-average actions, which are mismatched and vice-versa: for example, widely used normalization techniques like *min-max normalization* and *standardization* (i.e. zero-mean unit-variance normalization) lack this property. In particular, min-max normalization transforms values to be in range $[0, 1]$ such that the minimum value corresponds to 0 and the maximum to 1. Such normalization would make the normalized advantages to be always positive: thus, the sign is lost. Similarly, standardization would change the sign to be negative for those values which are below the mean value.

F. Reinforced Curriculum Learning

Since the problem of autonomous driving is extremely complex we adopt a stage-based learning procedure for our PPO agent, inspired by Curriculum Learning [12]. We divide

TABLE I

PERFORMANCE OF OUR AGENTS: *Curriculum* (C), *Standard* (S), AND *Untrained* (U). BEST RESULTS ARE HIGHLIGHTED IN BOLD. THE RESULTS HAVE BEEN AGGREGATED OVER THE TWO WEATHER SETS (*Soft* AND *Hard*), AND THREE TRAFFIC SCENARIOS (*No*, *Regular*, AND *Dense*)

Metric/ Town		Town01	Town02	Town03	Town04	Town05	Town06	Town07	Town10	Total
Collision rate	C	0.86	**0.78**	0.88	**0.51**	0.49	0.33	**0.77**	**0.48**	64%
	S	**0.79**	0.84	**0.7**	0.63	**0.4**	**0.3**	0.78	0.57	**63%**
	U	0.99	0.99	0.98	0.99	0.98	0.92	0.88	0.89	95%
Similarity	C	**0.95**	**0.95**	**0.94**	**0.92**	**0.91**	**0.96**	**0.89**	0.85	**92%**
	S	0.94	0.93	0.9	**0.92**	0.9	**0.96**	0.86	**0.9**	91%
	U	0.84	0.8	0.8	0.82	0.72	0.7	0.76	0.72	77%
Speed	C	7.78	**8.46**	8.13	**9.05**	8.55	**9.63**	7.65	**8.76**	8.5 km/h
	S	**8.58**	8.22	**8.43**	**9.05**	**9.36**	9.33	**7.68**	8.57	**8.65 km/h**
	U	5.96	5.7	6.04	5.98	6.38	6.55	5.75	6.25	6.08 km/h
Timesteps	C	296	**335**	347	**413**	406	468	**323**	**400**	**374**
	S	**316**	331	**371**	375	**428**	**471**	282	373	368
	U	191	207	237	207	268	313	215	269	238
Total reward	C	1866	**2530**	**2157**	**2161**	1764	1951	**1813**	**1961**	**2025**
	S	**2135**	2036	1996	1780	**2190**	**2030**	1479	1906	1944
	U	503	496	589	542	484	688	357	524	523
Waypoint distance	C	**1.54**	**1.44**	**1.75**	3.75	**3.74**	5.16	**2.18**	4.3	**2.98 m**
	S	1.77	1.97	2.98	3.9	3.8	4.69	2.24	**3.3**	3.08 m
	U	2.4	2.77	3.24	**3.2**	4.66	**4.43**	3.34	4.05	3.51 m

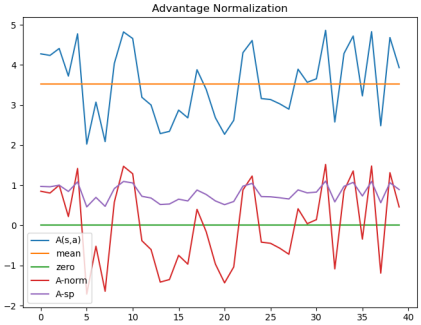


Fig. 5. How normalization alters the sign of the estimated advantages. The blue curve shows the true estimated advantages (notice they are all positive). Standardized advantages (red) became negative if they were below the average, compared to sign-preserved advantages (purple) which are still positive.

the whole reinforcement learning procedure into *five* stages of growing difficulty, such that the agent is guided to learn increasingly complex behavior. Each stage has a version of the learning environment ($\mathcal{E}$) that emphasizes specific aspects of the driving tasks. All the following stages occur in `Town 3`: one of the most complete and challenging town available in CARLA.

- **Stage 1**: the agent starting point is sampled from a fixed set of $n = 10$ locations (determined by fixing the random generator's seed). the agent's initial position is sampled from n = 10 positions. Also, in this situation the agent has to respect the speed limits and there are no other dynamic objects other than the one controlled by the agent (*no traffic* scenario).
- **Stage 2**: n is set to 50, to let the agent experience more diverse starting locations. In addition, the simulator tries to randomly generate a maximum of 50 pedestrians walking freely across the map, possibly crossing streets.
- **Stage 3**: there are no more restrictions on the start locations. Moreover, the weather condition is randomly set to one of the presets [`ClearNoon`, `ClearSunset`, `WetSunset`, `SoftRainSunset`, `CloudyNoon`, `WetNoon`, `SoftRainNoon`]. Finally, in addition to the 50 pedestrian, CARLA also inserts 50 vehicles into the map thus creating the *regular traffic* scenario.
- **Stage 4**: same conditions of Stage-3 but with *data-augmentation* enabled (as detailed in section IV-B).
- **Stage 5**: the *dense traffic* scenario is developed, 100 vehicles and 200 pedestrians are placed within the city. (Data augmentation is still enabled.)

Each stage lasts for 500 episodes of 512 timesteps, for a total of 1.28M timesteps.

V. RESULTS

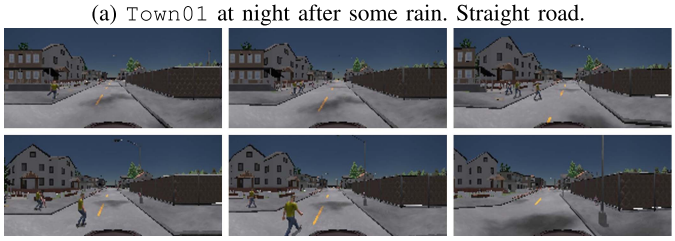
In this section we provide both quantitative (Table I) and qualitative (Fig. 6) results of the driving policy resulting from our reinforced curriculum learning procedure [45].

A. Evaluation Procedure

We perform extensive evaluation of our agent against six metrics, on *all* CARLA's towns with different weather conditions, as well as three traffic scenarios as proposed in the *NoCrash* benchmark [6]:

- **Metrics**: collision rate, similarity, waypoint distance, speed, total reward, and timesteps.
- **Towns**: in CARLA each town has its own unique features. We trained our agent only on *one* town, `Town03`, and evaluated it on `Town01`, `Town02`, `Town03`, `Town04`, `Town05`, `Town06`, `Town07`, `Town10`.
- **Weather**: we evaluate on two disjoint sets of weather presets. The first set (described in section IV-F) has been only used for training, the other is novel for the agent: [`WetCloudyNoon`, `WetCloudySunset`, `CloudySunset`, `HardRainNoon`, `MidRainyNoon`, `MidRainSunset`].
- **Traffic**: as in the *NoCrash* benchmark [6], we evaluate our agent on three different traffic scenarios: *no*

(a) `Town01` at night after some rain. Straight road.



(b) `Town03` at day. Pedestrian being avoided.

(c) `Town07` at sunset. Collision with a parked vehicle.

Fig. 6. Performance of our agent in various settings, towns and weather. Notice that scenario (a) and (c) are *novel*, not experienced by the agent during training.

traffic (without any pedestrian nor vehicle), *regular traffic* (50 pedestrians and 50 vehicles), and *dense traffic* (200 pedestrians and 100 vehicles).

We also evaluate the benefit of curriculum learning, comparing the same agent with and without curriculum: we refer to the former agent as *curriculum* (C), and the latter as *standard* (S). Moreover, we also provide (non-trivial) baseline performance of an agent with the same architecture as the other two but with random weights being *fixed* for the entire evaluation procedure: we refer to this agent as *untrained* (U). Notice that the untrained agent is a stronger (but still naive) baseline compared to a purely random-guess agent, which completely discards the input observations it receives solely sampling actions uniformly. Relative performance, aggregated over the three traffic scenarios as well as the two weather sets, are shown in Table I. Qualitative results are provided by Fig. 6.

B. Discussion

From the detailed evaluation results, we point out two major weaknesses of our approach: (1) the agent struggles at coordinating acceleration and breaking, and (2) at recognizing obstacles. This results in *low speed* (about 9 km/h) and *many collisions* as well. Such behavior could be due to lack of exploration, network capacity and/or architecture, as well as various difficulties in optimizing the policy gradient.

We also demonstrate the following: (1) emerging driving behavior without leveraging any domain knowledge, that is (2) *robust* and *consistent* across towns and weather conditions, furthermore (3) the stage-based reinforcement learning procedure has proven to be competitive, even better, compared to plain reinforcement learning.

VI. Conclusion

Deep reinforcement learning is still a relatively new field with lots of unexplored research directions, that enable us to solve even complex decision-making problems in a completely end-to-end fashion, thus without leveraging any domain-specific knowledge or expensive sets of highly-annotated data. On the contrary, imitation learning is a stronger approach for autonomous driving that heavily relies on high-quality and high-quantity datasets, which also should provide demonstrations of recovery from driving mistakes in order to learn a reliable driving policy.

Although our approach is not yet competitive with the state-of-the-art (CIRL [7], CAL [17], and CIRLS [6]), we demonstrate emerging driving behavior that is consistent across all CARLA towns and robust to change in weather. To our knowledge, we are the first to provide baseline performance on all towns, and to demonstrate such consistency. We also provide a decomposition of the returns that allows learning the value function in a stable and accurate way, as well as a proper normalization function for the estimated advantages.

References

[1] *International: On-Road Automated Vehicle Standards Committee*, S. SAE, Taxonomy Definitions Terms Rel. On-Road Motor Vehicle Automated Driving Syst., Warrendale, PA, USA, Inf. Rep., 2014.

[2] S. Pendleton *et al.*, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, Feb. 2017.

[3] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. Field Robot.*, vol. 37, no. 3, pp. 362–386, 2020.

[4] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[5] F. Codevilla, M. Müller, A. Lopez, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2018, pp. 1–9.

[6] F. Codevilla, E. Santana, A. Lopez, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9329–9338.

[7] X. Liang, T. Wang, L. Yang, and E. Xing, "CIRL: Controllable imitative reinforcement learning for vision-based self-driving," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 584–599.

[8] A. Kendall *et al.*, "Learning to drive in a day," in *Proc. Int. Conf. Robot. Automat. (ICRA)*, May 2019, pp. 8248–8254.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," 2017, *arXiv:1711.03938*.

[10] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, M. Hutter and R. Siegwart, Eds. Cham, Switzerland: Springer, 2018, pp. 621–635.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[12] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 41–48.

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

[14] S. Hecker, D. Dai, and L. Van Gool, "End-to-end learning of driving models with surround-view cameras and route planners," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 435–453.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[17] A. Sauer, N. Savinov, and A. Geiger, "Conditional affordance learning for driving in urban environments," 2018, *arXiv:1806.06498*.

[18] W. Zeng *et al.*, "End-to-end interpretable neural motion planner," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 8660–8669.

[19] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.

[21] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.

[22] J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," in *Proc. IEEE Intell. Transp. Syst. Conf. (ITSC)*, Oct. 2019, pp. 2765–2771.

[23] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," 2015, *arXiv:1509.06461*.

[24] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018, *arXiv:1802.09477*.

[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, *arXiv:1801.01290*.

[26] C. Li and K. Czarnecki, "Urban driving with multi-objective deep reinforcement learning," 2018, *arXiv:1811.08586*.

[27] P. A. Lopez *et al.*, "Microscopic traffic simulation using sumo," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Nov. 2018, pp. 2575–2582.

[28] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, (2000). *TORCS, The Open Racing Car Simulator*. [Online]. Available: http://torcs.sourceforge.net

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[30] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.

[31] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1, no. 2. Cambridge, MA, USA: MIT Press, 2016.

[32] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 834–843.

[33] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[34] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[36] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[37] G. Brockman *et al.*, "OpenAI gym," 2016, *arXiv:1606.01540*.

[38] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[39] F. Codevilla, A. M. Lopez, V. Koltun, and A. Dosovitskiy, "On offline evaluation of vision-based driving models," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 236–251.

[40] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*.

[41] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 116–131.

[42] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Netw.*, vol. 107, pp. 3–11, Nov. 2018.

[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*.

[44] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[45] L. Anzalone, S. Barra, and M. Nappi, "Reinforced curriculum learning for autonomous driving in CARLA," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2021, pp. 3318–3322.

Paola Barra received the B.S. degree in computer science from the University of Salerno, the M.S. degree in business informatics from the University of Pisa, the Ph.D. degree from the Univerisity of Salerno in 2021. Her research interests include machine learning techniques to solve issues using computer, as vision facial and gait recognition, action recognition, and tumor detection. She is a member of GIRPR/IAPR.

Silvio Barra was born in Battipaglia, Salerno, Italy, in 1985. He received the B.Sc. and M.Sc. degrees *(cum laude)* in computer science from the University of Salerno, in 2009 and 2012, respectively, and the Ph.D. degree from the University of Cagliari, in 2017. Currently, he is a Research Assistant with the University of Naples, Federico II. He has authored more than 50 papers, published in international journals, conferences, and books. His main research interests include pattern recognition, biometrics, video analysis and analytics, and financial forecasting.

Aniello Castiglione (Member, IEEE) received the Ph.D. degree in computer science from the University of Salerno, Italy. He is currently an Associate Professor with the University of Naples Parthenope, Italy. He received the Italian National Qualification as a Full Professor of computer science in 2021. He published more than 240 papers in international journals and conferences. Considering his journal articles, more than 85 of them are ranked Q1 in Scopus/Scimago classification and more than 70 of them are ranked Q1 in the Clarivate Analytics/ISI-WoS classification. The international academic profile of him is spread among his 86 international coauthors who belong to 75 different institutions located in 18 countries. He served in the organizations as the Program Chair and a TPC Member in around 250 international conferences (some of them are ranked A+/A/A- in the CORE, LiveSHINE, and Microsoft Academic international classifications). His current research interests include information forensics, digital forensics, security and privacy on cloud, communication networks, applied cryptography, and sustainable computing. Currently, he is the Editor-in-Chief of the Special Issues for the *Journal of Ambient Intelligence and Humanized Computing* (Springer). He served as the Managing Editor for two ISI-ranked international journals and as a Reviewer for 110 international journals. In addition, he served as a Guest Editor for 30 Special Issues and served as the Editor of more than ten Editorial Boards of international journals, such as IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING, IEEE ACCESS, *IET Image Processing* (IET), *Journal of Ambient Intelligence and Humanized Computing* (Springer), *MTAP*, *Sustainability* (MDPI), *Smart Cities* (MDPI), and *Future Internet* (MDPI). One of his papers (published in the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING) was selected as "Featured Article" in the "IEEE Cybersecurity Initiative" in 2014. In October 2020 and October 2021, he was included into the ranking of the top 100,000 scientists for the years 2019 and 2020. He is a member of ACM.

Michele Nappi (Senior Member, IEEE) received the laurea degree *(cum laude)* in computer science from the University of Salerno, Italy, in 1991, the M.Sc. degree in information and communication technology from I.I.A.S.S. E.R. Caianiello, in 1997, and the Ph.D. degree in applied mathematics and computer science from the University of Padova, Italy, in 1997. He was one of the founders of the spin off BS3 (biometric system for security and safety) in 2014. He is currently a Full Professor of computer science with the University of Salerno. He is a Team Leader of the Biometric and Image Processing Laboratory (BIPLAB). He has authored more than 180 papers in peer-reviewed international journals, international conferences, and book chapters. His research interests include pattern recognition, image processing, image compression and indexing, multimedia databases and biometrics, human–computer interaction, and VR/AR. He is a member of TPC of international conferences. He is a GIRPR/IAPR Member. He received several international awards for scientific and research activities. He is the Co-Editor of several international books. He serves as an Associate Editor and a Managing Guest Editor for several international journals. He is the President of the Italian Chapter of the IEEE Biometrics Council.

Luca Anzalone received the B.Sc. and M.Sc. degrees *(cum laude)* in computer science from the University of Salerno, in 2018 and 2020, respectively. He is currently pursuing the Ph.D. degree in data science and computation with the University of Bologna. His research interests include deep learning and deep reinforcement learning.