Simulation-based reinforcement learning for autonomous driving

Christopher Galias [*1 2] **Adam Jakubowski** [*1] **Henryk Michalewski** [*1 3] **Błażej Osiński** [*1 3] **Paweł Zięcina** [*1]

Abstract

We use synthetic data and a reinforcement learning algorithm to train a driving policy intended to control steering of a full-size real-world vehicle in a number of restricted driving scenarios. The driving policy uses RGB images as input.

1. Introduction

The premise of this work is to verify whether a large number of synthetic samples is enough to train a useful end-to-end driving policy which takes RGB images as input and outputs continuous steering commands. The experiment is supported by one of the leading car manufacturers, which provides an appropriately instrumented full-sized passenger vehicle for testing.

The experiment is designed with mostly business considerations in mind. In principle we may use real-world data, but it must be already available or cheap to obtain. The driving policy is evaluated only with regard to its real-world performance at the speed of 30km per hour. Real-world scenarios are limited to a number of fixed routes outlined in Section 3. In order to complete a scenario the driving agent is expected to execute from approximately 1000 to 8000 actions at 5 Hz. For simplicity, in real-world and simulated scenarios we are allowing only static objects. The input provided by the real car is limited to an RGB image.

In this initial series of experiments we have decided to limit intermediate human-designed or learned representations of the real world only to semantic segmentation. Our driving policies are trained directly on visual inputs, understood as RGB images along with their segmentations and high-level commands described in Section 3 and inspired by (Codevilla et al., 2017).

Controversial at first glance, the decision of training a reinforcement learning policy directly on pixels was motivated by a practical observation made by our team in experiment (Kaiser et al., 2019), in which we have observed that learning world models of Atari games is a difficult task. Since learning good representations of environments described in Section 3 may be potentially even harder, we have decided to base our initial experiments on pixel inputs. This decision, in turn, limited the number of applicable reinforcement learning algorithms to a subfamily of actor-critic algorithms known for state-of-the-art performance on RGB inputs (see Section 4 for further discussion). Reward shaping is allowed in our experiments, but we made an effort to use as simple reward functions as possible (described in Section 3.2).

As stated in Section 5 of (Dosovitskiy et al., 2017): "when using a realistic simulator, an extensive hyperparameters search becomes infeasible". We have decided to use the same realistic simulator as in (Dosovitskiy et al., 2017) — CARLA, based on Unreal Engine 4, and overcome the infeasibility with a parallelized training architecture inspired by IMPALA (Espeholt et al., 2018), Ape-X (Horgan et al., 2018), OpenAI Five (OpenAI et al., 2018), Horovod (Sergeev & Del Balso, 2018), and DBA3C (Adamski et al., 2018). Details of our parallelization approach are presented in Section 5.4. With our current infrastructure and parallelization methods in this experiment we are planning to generate in total 100 years of simulated driving experience.

Because of business and legal considerations and certain hardware limitations not all details of our experiments are reported in this paper with full scientific rigor. In particular in this release of our work we are able to provide only qualitative description of real-world performance of our policies. In the future we will add more quantitative details with regard to real-world deployments along with appropriate ablations. Section 5 contains experiments which can be presented at this stage. Additional comments and videos are accessible via a dedicated webpage: `http://bit.ly/2JiQ9Ur`.

2. Related work

Synthetic data and real-world robotics Synthetic images were used by Pomerleau (1988) in the ALVINN experiment. Sadeghi & Levine (2016) proposed a training procedure for drones and (James et al., 2017; Pinto et al., 2018; Peng et al., 2018; Tobin et al., 2018; OpenAI et al., 2018) proposed experiments with robotic manipulators where

[*]Equal contribution [1]deepsense.ai [2]Jagiellonian University [3]University of Warsaw. Correspondence to: Christopher Galias <chris.galias@gmail.com>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

training was performed using only synthetic data. Progressive nets and data generated using the MuJoCo engine (Todorov et al., 2012) were used by Rusu et al. (2016) to learn policies in the domain of robot manipulation. A driving policy for a one-person vehicle was trained by Bewley et al. (2018). The policy is reported to show good performance on a rural road and the training used mostly synthetic data generated by Unreal Engine 4. Our inclusion of segmentation as described in Section 3.5 is inspired by sim2real experiments presented in (James et al., 2018). Visual servoing systems inspired by (Sadeghi & Levine, 2016) and trained using synthetic data were presented in (Sadeghi et al., 2017; Sadeghi, 2019).

Synthetic data and simulated robotics Emergence of high-quality general purpose physics engines such as MuJoCo (Todorov et al., 2012), along with game engines such as Unreal Engine 4 and Unity, and their specialized extensions such as CARLA (Dosovitskiy et al., 2017) or AirSim (Shah et al., 2017), allowed for creation of sophisticated photo-realistic environments which can be used for training and testing of autonomous vehicles. A deep reinforcement learning framework for autonomous driving was proposed by Sallab, Abdou, Perot, and Yogamani (2017) and tested using the racing car simulator TORCS.

Reinforcement learning methods led to very good performance in simulated robotics, see for example solutions to complicated walking tasks in Heess et al. (2017); Kidzinski et al. (2018). In the context of CARLA, impressive driving policies were trained using imitation learning (Codevilla et al., 2017; Rhinehart et al., 2018b), affordance learning (Sauer et al., 2018), reinforcement learning (Chen et al., 2019), and a combination of model-based and imitation learning methods proposed by Rhinehart, McAllister, and Levine (2018a). However, as Bewley et al. (2018) state: "training and evaluating methods purely in simulation is often 'doomed to succeed' at the desired task in a simulated environment" and indeed, in our suite of experiments described in Section 3 most of simulated tasks can be relatively easily solved, in particular when a given environment is deterministic and simulated observations are not perturbed.

Reinforcement learning and real-world robotics An extensive survey of various applications of reinforcement learning in robotics can be found in Deisenroth et al. (2013, Section 2.5). The role of simulators and reinforcement learning is discussed by Sünderhauf et al. (2018) in Section IV. In Sadeghi & Levine (2016); Pinto et al. (2018); Peng et al. (2018); Tobin et al. (2018); OpenAI et al. (2018); James et al. (2018); Bewley et al. (2018); Rusu et al. (2016) policies are deployed on real-world robots and training is performed mostly using data generated by simulators. Kang, Belkhale, Kahn, Abbeel, and Levine (2019) propose a system where dynamics are trained using real-world data and perception is trained using synthetic data. Training of a deep reinforcement learning policy using the TORCS engine and deployment on real-world data are presented in a recent work by Tan, Xu, and Kong (2018).

3. Environment

3.1. Simulator

We use CARLA (Dosovitskiy et al., 2017),[1] an open-source simulator for autonomous driving research based on Unreal Engine 4. CARLA features open assets (including two built-in maps), 14 different weather settings, semantic segmentation for the assets, as well as camera and LIDAR sensors (in our experiments we only use RGB information). Camera position, orientation and settings are customizable. CARLA also features multiple vehicles with different physical parameters. Two visual quality levels (`LOW` and `EPIC`) are supported.[2]

We have also recreated a real-world urban space. The space is presented as two new CARLA maps which approximately reflect the testing grounds for real-world deployments. We use these maps along with maps provided in CARLA for training, with a number of scenarios reserved for validation only.

3.2. Scenarios and rewards

Our agent is trained on a number of fixed routes. Each route is a separate training scenario. Scenarios can be arbitrarily mixed in a training procedure. In the current stage of the experiment the scenarios do not include any dynamic obstacles. The number of timesteps for a given scenario ranges from $\sim$1000 to $\sim$8000. Agents are expected to drive on their own lanes, but other traffic rules are ignored. See Figure 1 for an example route.

The goal of the agent in a given scenario is to drive along a route from start to finish. The route is defined by a list of checkpoints on the map. The agent receives reward in proportion to its progress along the polygonal path spanned by the checkpoints (see Figure 2), i.e.,

$$m_t = \mathrm{proj}(\mathrm{pos_t}) - \mathrm{proj}(\mathrm{pos_{t-1}}),$$

where $\mathrm{pos_t}$ is the position of the agent at timestep t and where proj() denotes the orthogonal projection onto the polygonal path spanned by the checkpoints[3] (see Figure 2).

The agent also receives a center-line penalty for driving

[1] Version `0.8.4`.

[2] The `EPIC` quality level adds shadows, water reflections, sun flare effect, and antialiasing.

[3] When a given point's projection falls outside of a given line segment, it gets projected to the closer endpoint.

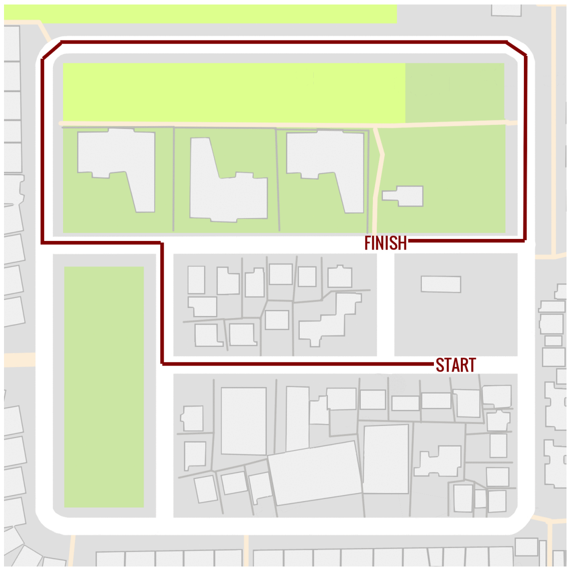

Figure 1. Example route on CARLA built-in Town2 level. We use few scenarios per each CARLA level and a number of scenarios defined on our custom maps.

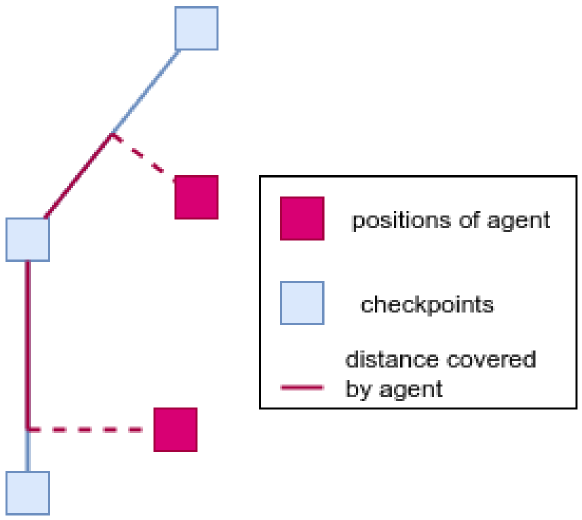

Figure 2. Reward calculation.

outside of a fixed-width band centered on the path, i.e.,

$$c_t = \max(\|\mathrm{pos_t} - \mathrm{proj}(\mathrm{pos_t})\| - \mathrm{width}/2, 0),$$

where $\|\cdot\|$ is the euclidean norm and width is the road width. The whole reward at timestep t then is

$$r_t = m_t + \alpha c_t,$$

where α is a hyperparameter set to 0.1. Furthermore, the episode ends if the agent strays too far from the route, i.e., if

$$\|\mathrm{pos_t} - \mathrm{proj}(\mathrm{pos_t})\| \geq 10.$$

3.3. Actions

In our scenarios vehicles are controlled by two values — throttle and steering. Our policy controls only the steering. Throttle is controlled via a PID controller with the speed set to a constant.

Even though expressing steering as a discrete value might lead to easier training, in the current stage of the experiment we decided to use continuous values in order to avoid a somewhat arbitrary discretization step.

We model continuous actions with the Gaussian distribution. Since the Gaussian distribution is unimodal, it cannot model bimodal situations like `turn either left or right`. This is not a problem since we are passing an explicit high-level navigation command to the policy described in detail in Section 3.4.

We report some of our findings on modelling continuous actions in section 5.2.

3.4. Observations

The agent receives as its observation an RGB image from a single front camera and car metrics such as speed and acceleration. The camera position and orientation was configured to mimic the real-world setup.

The agent is also provided with high-level navigation command, which can be one of the following: `LANE FOLLOW`, `GO STRAIGHT`, `TURN RIGHT`, `TURN LEFT`. As there is inherent ambiguity in the environment (for example, at intersections, where both turning left and turning right are reasonable actions), this additional command is a convenient design decision which does not conflict with practical applications (see (Codevilla et al., 2017) for a broader discussion).

3.5. Semantic segmentation

In our standard experiment the input to the policy consists of an RGB camera image concatenated with a semantic segmentation mask. The semantic segmentation model is trained in a supervised way outside of the reinforcement learning loop. For our semantic segmentation model we used the U-Net (Ronneberger et al., 2015) architecture. We trained the semantic segmentation model on synthetic data from CARLA and real-world labelled data provided by our industry partner.

3.6. Domain randomization

Domain randomization is a technique aimed to improve generalization of the learned policy. Similarly to many other sim2real experiments discussed in Section 2, it is done via randomizing different aspects of observations and is intended to prevent overfitting to idiosyncrasies of the simulation. At the beginning of each episode we sample the following parameters of the environment:

- the weather (out of 14 weather settings),
- the scenario,
- camera position, orientation and field of view,
- camera image brightness, blur, and noise,
- parameters of cutout patches,[4]
- parameters of car dynamics.

We apply some randomizations in a supervised manner to improve sample efficiency. For more details see Section 5.3.

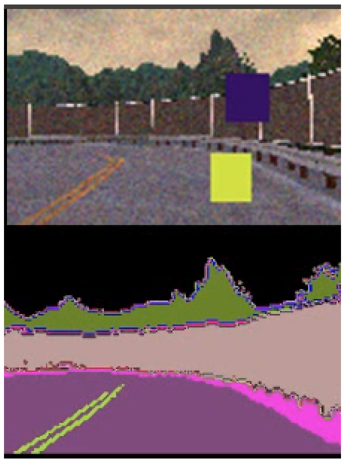

Figure 3. Top: RGB camera input with cutout randomization. Bottom: semantic segmentation predictions.

4. Main algorithm

The main training loop is described in Algorithm 1. As the reinforcement learning component we use Proximal Policy Optimization (PPO) (Schulman et al., 2017) with a continuous action space. A good alternative to PPO would be TRPO (Schulman et al., 2015) or V-trace (Espeholt et al., 2018), which we leave for future work. Supervised randomizations mentioned in line 5 of Algorithm 1 and described in Section 5.3 may be considered as a potentially more sample-efficient variant of general domain randomizations.

We based our PPO implementation on OpenAI Baselines (Dhariwal et al., 2017) `ppo2`[5]. Policies provided by OpenAI Baselines work only for observations that consists of one tensor of some shape. In our case we operate on multiple tensors of multiple shapes — such us the front camera, a high-level navigation command, car speed, and car acceleration. We use a custom policy that operates on multiple input tensors coming from one observation (see Figure 4).

[4]Occluding parts of the image (see Figure 3 and (DeVries & Taylor, 2017)).

[5]`https://github.com/openai/baselines/tree/master/baselines/ppo2`

Algorithm 1 Training loop.

```
1: initialize policy $\pi$
2: for $i = 1$ to num_epochs do
3:     collect data using policy $\pi$
4:     calculate $\text{loss}_{\text{ac}}$ using an actor-critic algorithm
5:     calculate $\text{loss}_{\text{sr}}$ using supervised randomizations
6:     $\text{loss} = \text{loss}_{\text{ac}} + \text{loss}_{\text{sr}}$
7:     update $\pi$ using $\nabla\text{loss}$
8: end for
```

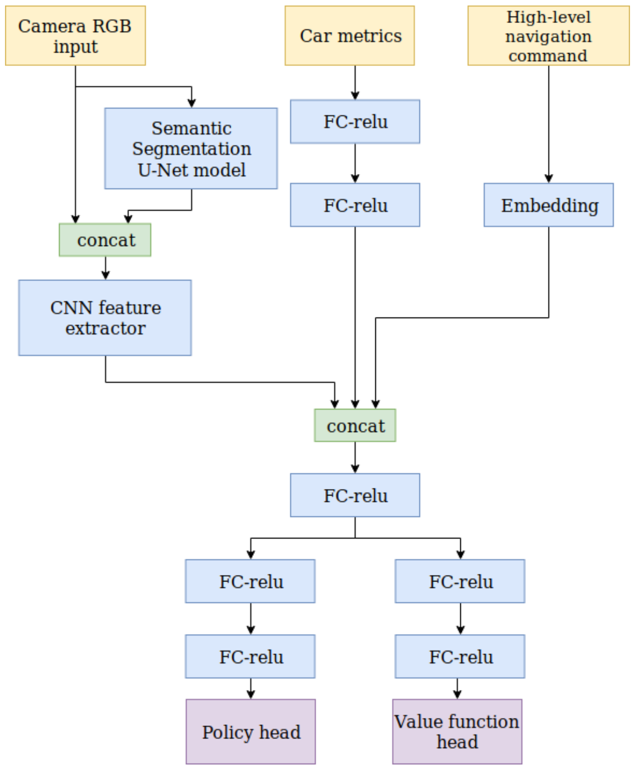

Figure 4. Network architecture.

5. Experiments

In Section 5.1 we describe saliency map generation, which proved to be useful tool in debugging our RL pipeline.

In Section 5.2 we discuss different ways of modeling the standard deviation parameter of the action distribution, which turned out to be crucial when working with a continuous action space.

In Section 5.3 we describe domain randomizations applied in a supervised manner, which allows for better sample efficiency.

In section 5.4 we discuss our distributed training setup. Finally, in Section 5.5 we describe various failure cases.

5.1. Saliency maps

In order to gain some insight into the inner workings of the policy we generate saliency maps that visualize policy output sensitivity to different parts of camera input. Saliency maps were introduced early in the project as a way to understand some results when deploying policies in real life (example videos from training with saliency maps can be seen in complementary materials at `http://bit.ly/2ZYkRYH`).

The first and simplest way to generate saliency map is to utilize the calculation graph framework and calculate the gradient of the output with respect to policy input. As said in (Greydanus et al., 2017): "The simplest approach is to take the Jacobian with respect to the output of interest. Unfortunately, the Jacobian does not usually produce human-interpretable saliency maps". Such saliency maps are very noisy on a single-pixel level. Results can be seen on the left image in Figure 5.

Greydanus et al. (2017) proposed an alternative way to calculate saliency maps. Their technique consists of blurring different patches of the input image (effectively removing some information from that patch) and measuring the output difference. This second technique generated more easily interpretable maps, but at the cost of generation time. The policy network needs to be applied hundreds of times to different perturbations of one image to generate the saliency maps. In our case, generating a video with saliency maps for a 17-minute piece of footage took about $\sim$2 hours on a machine with a GPU.

Finally, we decided to use a hybrid approach — calculating the gradient analytically, like in first approach, but using patches instead of single pixels, like in second approach. This results in a saliency map of acceptable quality with a fast generation time (a few minutes for a 17-minute piece of footage). An example can be seen on Figure 5.

5.2. Modelling standard deviation of a continuous action distribution

The Gaussian distribution is described by two parameters — the mean and the standard deviation. There are multiple ways how to handle these parameters in the policy. Using some approaches over the other turned out to be quite relevant for the performance of agents. In this section we will describe tried approaches and their results.

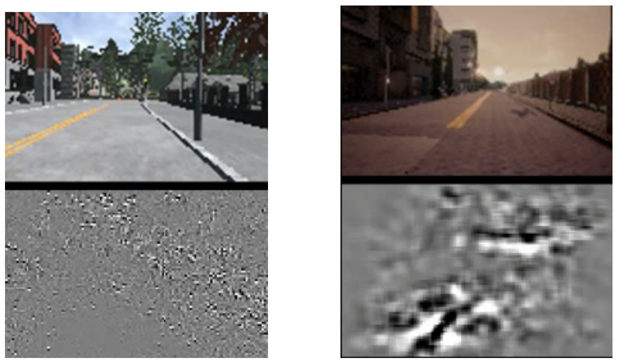

Figure 5. Left: Saliency map generated w.r.t. single pixels. Right: Saliency map generated w.r.t. 5×5 pixel patches. Each pixel encodes output sensitivity with regard to some input patch around it. White denotes positive gradient and black denotes negative gradient. Gray denotes gradient values close to zero. The saliency maps show that the network is most sensitive to road curbs and lane markings.

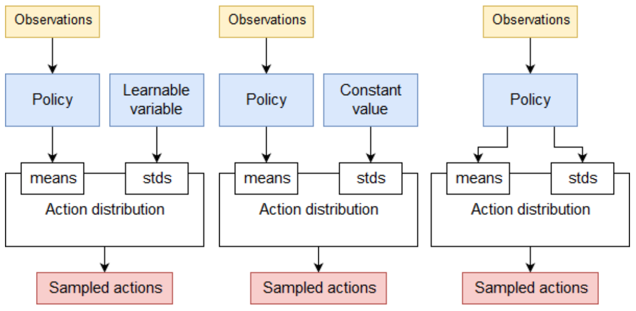

Figure 6. Different ways of modelling standard deviation.

5.2.1. Learnable standard deviation value detached from the policy

In our first approach the mean of the distribution was defined as the ouput of the policy network and the standard deviation was provided by a global learnable variable (see the left of figure 6), an approach used in the implementation of the PPO algorithm presented in OpenAI Baselines.

Ideally, we would like to approach a deterministic policy at convergence (the agent should drive steadily without any exploratory movements). With the above approach we noticed that the policy entropy never goes down below a certain point and the agent does not drive steadily. Furthermore, even when initializing the standard deviation with a small value the policy entropy still climbed back up. In our experience such policies perform poorly when deployed on real-world cars.

The simulated (and real-world) car has its inertia: while the policy can output any value for requested steering, it still takes some time for the car steering wheel to respond. Additionally, there is a limit on how much the steering wheel can rotate in one simulation step. If policy actions change too rapidly between successive observations they effectively get truncated by inertia (see Figure 7). We hypothesize that this causes the phenomenon of entropy not going below a certain point.

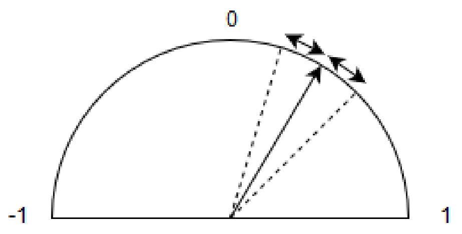

Figure 7. The policy can output any action in $[-1, 1]$, but there is a limit on how much the steering wheel can physically rotate during one step.

Another issue is that different parts of scenarios have different characteristics when it comes to exploration. Some parts of scenarios (e.g. driving straight) are relatively simple, whereas others (like intersections) require more exploration to solve.

If we utilize one shared value for standard deviation for all observations then those different modes compete with each other during optimization.

An example video can be seen at `http://bit.ly/2vG1mpQ`.

5.2.2. Small constant standard deviation

In our second approach the mean of the distribution was output by the policy network (as previously), but the standard deviation was set to a small constant value (see the center of figure 6). Because our reward function is not sparse the amount of exploration was still sufficient for the policy to solve the scenarios. Furthermore, because the standard deviation is small the policy never suffers from the aforementioned truncation by the environment inertia.

The policy trained this way performed much better when deployed on a real-world car. It handled reliably wide-road scenarios, but struggled with narrow-road scenarios. While the standard deviation was very small, it was still not small enough to work decently in very precise scenarios like narrow roads.

An example video can be seen at `http://bit.ly/2ZZI7Wk`.

5.2.3. Policy outputs both mean and standard deviation

The most successful approach had the policy controlling both the mean and the standard deviation (see the right of figure 6). This approach also worked the best when deployed on real-world car. The driving was steady and it handled more narrow scenarios that previous approaches struggled with.

This approach allows the policy to adjust the degree of exploration on a per-observation basis. In the supplementary video material one can see an example of less exploration in easy and solved straight sections of the scenario and more exploration in harder and not-yet-mastered parts of the scenario, such as intersections.

An important implementation detail was to enforce an upper boundary for the standard deviation. Without such a boundary the standard deviation would sometime *explode* and never go down below a certain point, similarly to the first approach (see section 5.2.1).

An example video can be seen at `http://bit.ly/2VI6uZn`.

5.3. Supervised randomizations

This section describes a supervised approach to domain randomization, first mentioned in in Algorithm 1.

As each randomization introduces more variance to the observations, the amount of samples needed to estimate the policy gradient increases rapidly. As (OpenAI et al., 2018) write:

“Learning to rotate an object in simulation without randomizations requires about 3 years of simulated experience, while achieving the same performance in a fully randomized simulation requires about 100 years of experience”.

Some of our randomizations are applied only to visual camera input (blur, cutout, noise, camera positions, camera FOV) and some to car dynamics (car controller and physics parameters).

One way of sidestepping the aforementioned sample efficiency issue is moving the visual randomizations outside of the policy gradient. This can be done as follows:

1. During policy rollouts, the network is fed with unperturbed visual input.
2. For each observation in the collected data batch a copy with perturbed[6] camera image is prepared.
3. We introduce an additional loss that conditions the

[6]We used the augmentation library (A. Buslaev & Kalinin, 2018) to perturb images.

policy to output a similar distribution for the perturbed and unperturbed images (line 5 in algorithm 1).

For this purpose we use the *mean absolute error* (MAE) between the means of the distributions output by the agent, i.e.,

$$\mathrm{MAE}(\pi(\mathrm{obs}).\mathrm{mean}, \pi(\mathrm{obs}_{\mathrm{perturbed}}).\mathrm{mean}),$$

where π is the policy network outputting the mean and standard deviation parameters, and obs is the input to the agent. We have also experimented with different distribution similarity measures, but MAE on distribution means worked well enough in practice.

Heuristically, MAE seems a better choice than *mean squared error*, as in the case of multimodality, we'd prefer the output to collapse to one of the modes instead of taking their average.

5.4. Parallelization and performance

We are training our agents using Horovod (Sergeev & Del Balso, 2018), a distributed framework which allowed for smooth integration with our main computation resource. Our typical setup consists of running experiments on up to 8 nodes each equipped with 2 GPUs. On each node we run the CARLA simulators on the first GPU and the training code on the second GPU. Usually, we run 10 CARLA simulations per node. For the experiments of our scale we observe linear scaling in in the multi-node setup. FPS can vary a lot depending on settings of an experiment, see Table 1.

We are using a homogeneous setup where each node is running the same process (see Figure 8) — both policy rollouts and policy updating is done on all nodes. Gradients are averaged between nodes via Horovod. An interesting, but more demanding solution would be a master-worker setup used, for example, in (Espeholt et al., 2018; Adamski et al., 2018).

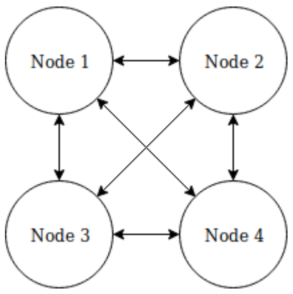

Figure 8. All nodes execute the same code. Gradients are averaged between nodes via Horovod.

[7]The quality level is sampled uniformly from {LOW, EPIC} on a per episode basis.

Table 1. FPS per node with various settings

SETTINGS	FPS
LOW QUALITY	140
LOW/EPIC[7] QUALITY	60
LOW/EPIC[7] QUALITY + SUPERVISED RANDOMIZATION	20

5.5. Selected failure cases

5.5.1. PULSE-WIDTH MODULATION STEERING STRATEGY

In one of the experiments we provided the policy with information of the last action. Intuitively, introducing such an autoregressive feature seems very useful. At the time CARLA did not expose information about exact in-simulation position of the steering wheel, hence we used the last action as a proxy for actual in-simulation position. As mentioned in Section 5.2, the steering wheel has its inertia.

The trained policy took advantage of this fact and was rapidly switching between two modes: extreme left and extreme right, which amounts to driving forward (see the video at http://bit.ly/2H4OWNQ).

5.5.2. SINGLE-LINE VERSUS DOUBLE-LINE ROAD MARKINGS

Our first trained policies were trained on CARLA levels which feature only double-line road markings. When evaluated on real-world footage, such a policy was not sensitive to single-line road markings, whereas it was sensitive to double-line road markings in simulation.

This problem was fixed was fixed after introducing our custom CARLA level which features double-line road markings. For visualizations see our complementary site at http://bit.ly/2DNWsfa.

6. Conclusions and future work

We have presented a qualitative overview of a series of experiments intended to train an end-to-end driving policy using the CARLA simulator. Policies were deployed on a full-size car.

In a future release of this work we will provide quantitative analysis of our experiments, including ablations. We are also planning to include results on memory-augmented policies, the V-trace algorithm (Espeholt et al., 2018), asymmetric actor-critic architecture resented by Pinto et al. (2018), and our own generator-discriminator pair inspired by (Bousmalis et al., 2017). Another interesting and challenging direction is integration of an intermediate representation layer — for example a 2D-map or a bird's-eye view, as proposed in

(Chen et al., 2019; Rhinehart et al., 2018a; Djuric et al., 2018; Bansal et al., 2018). It would be also interesting to focus reinforcement learning training on part of scenarios with the highest uncertainty, see e.g. (Kendall et al., 2017). Integration of a model-based method similar to presented by Lowrey et al. (2018) would be a desirable step towards better sample efficiency.

7. Acknowledgments

This was work was supported by the Academic Computer Center Cyfronet at the AGH University of Science and Technology in Kraków, Poland.

References

A. Buslaev, A. Parinov, E. K. V. I. I. and Kalinin, A. A. Albumentations: fast and flexible image augmentations. *ArXiv e-prints*, 2018.

Adamski, I., Adamski, R., Grel, T., Jedrych, A., Kaczmarek, K., and Michalewski, H. Distributed deep reinforcement learning: Learn how to play atari games in 21 minutes. *CoRR*, abs/1801.02852, 2018. URL http://arxiv.org/abs/1801.02852.

Bansal, M., Krizhevsky, A., and Ogale, A. S. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *CoRR*, abs/1812.03079, 2018. URL http://arxiv.org/abs/1812.03079.

Bewley, A., Rigley, J., Liu, Y., Hawke, J., Shen, R., Lam, V., and Kendall, A. Learning to drive from simulation without real world labels. *CoRR*, abs/1812.03823, 2018. URL http://arxiv.org/abs/1812.03823.

Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *CoRR*, abs/1709.07857, 2017. URL http://arxiv.org/abs/1709.07857.

Chen, J., Yuan, B., and Tomizuka, M. Model-free deep reinforcement learning for urban autonomous driving. *CoRR*, abs/1904.09503, 2019. URL http://arxiv.org/abs/1904.09503.

Codevilla, F., Müller, M., Dosovitskiy, A., López, A., and Koltun, V. End-to-end driving via conditional imitation learning. *CoRR*, abs/1710.02410, 2017. URL http://arxiv.org/abs/1710.02410.

Deisenroth, M. P., Neumann, G., and Peters, J. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013. doi: 10.1561/2300000021. URL https://doi.org/10.1561/2300000021.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. OpenAI Baselines. https://github.com/openai/baselines, 2017.

Djuric, N., Radosavljevic, V., Cui, H., Nguyen, T., Chou, F., Lin, T., and Schneider, J. Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *CoRR*, abs/1808.05819, 2018. URL http://arxiv.org/abs/1808.05819.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL http://arxiv.org/abs/1802.01561.

Greydanus, S., Koul, A., Dodge, J., and Fern, A. Visualizing and understanding atari agents, 2017.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M. A., and Silver, D. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017. URL http://arxiv.org/abs/1707.02286.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., and Silver, D. Distributed prioritized experience replay. *CoRR*, abs/1803.00933, 2018. URL http://arxiv.org/abs/1803.00933.

James, S., Davison, A. J., and Johns, E. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *CoRR*, abs/1707.02267, 2017. URL http://arxiv.org/abs/1707.02267.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *CoRR*, abs/1812.07252, 2018. URL http://arxiv.org/abs/1812.07252.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G., and Michalewski, H. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019. URL http://arxiv.org/abs/1903.00374.

Kang, K., Belkhale, S., Kahn, G., Abbeel, P., and Levine, S. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. *CoRR*, abs/1902.03701, 2019. URL http://arxiv.org/abs/1902.03701.

Kendall, A., Badrinarayanan, V., and Cipolla, R. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. In *British Machine Vision Conference 2017, BMVC 2017, London, UK, September 4-7, 2017*, 2017. URL https://www.dropbox.com/s/jgozsaobbk98azy/0205.pdf?dl=1.

Kidzinski, L., Mohanty, S. P., Ong, C. F., Huang, Z., Zhou, S., Pechenko, A., Stelmaszczyk, A., Jarosik, P., Pavlov, M., Kolesnikov, S., Plis, S. M., Chen, Z., Zhang, Z., Chen, J., Shi, J., Zheng, Z., Yuan, C., Lin, Z., Michalewski, H., Milos, P., Osinski, B., Melnik, A., Schilling, M., Ritter, H. J., Carroll, S. F., Hicks, J. L., Levine, S., Salathé, M., and Delp, S. L. Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. *CoRR*, abs/1804.00361, 2018. URL http://arxiv.org/abs/1804.00361.

Lowrey, K., Rajeswaran, A., Kakade, S., Todorov, E., and Mordatch, I. Plan online, learn offline: Efficient learning and exploration via model-based control. *CoRR*, abs/1811.01848, 2018. URL http://arxiv.org/abs/1811.01848.

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pp. 1–8, 2018. doi: 10.1109/ICRA.2018.8460528. URL https://doi.org/10.1109/ICRA.2018.8460528.

Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., and Abbeel, P. Asymmetric actor critic for image-based robot learning. In *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018. doi: 10.15607/RSS.2018.XIV.008. URL http://www.roboticsproceedings.org/rss14/p08.html.

Pomerleau, D. ALVINN: an autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, pp. 305–313, 1988.

Rhinehart, N., McAllister, R., and Levine, S. Deep imitative models for flexible inference, planning, and control. *CoRR*, abs/1810.06544, 2018a. URL http://arxiv.org/abs/1810.06544.

Rhinehart, N., McAllister, R., and Levine, S. Deep imitative models for flexible inference, planning, and control. *CoRR*, abs/1810.06544, 2018b. URL http://arxiv.org/abs/1810.06544.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation, 2015.

Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. Sim-to-real robot learning from pixels with progressive nets. *CoRR*, abs/1610.04286, 2016. URL http://arxiv.org/abs/1610.04286.

Sadeghi, F. Divis: Domain invariant visual servoing for collision-free goal reaching. *CoRR*, abs/1902.05947, 2019. URL http://arxiv.org/abs/1902.05947.

Sadeghi, F. and Levine, S. (cad)^{2}rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016. URL http://arxiv.org/abs/1611.04201.

Sadeghi, F., Toshev, A., Jang, E., and Levine, S. Sim2real view invariant visual servoing by recurrent control. *CoRR*, abs/1712.07642, 2017. URL http://arxiv.org/abs/1712.07642.

Sallab, A. E., Abdou, M., Perot, E., and Yogamani, S. Deep reinforcement learning framework for autonomous driving. *CoRR*, abs/1704.02532, 2017. URL http://arxiv.org/abs/1704.02532.

Sauer, A., Savinov, N., and Geiger, A. Conditional affordance learning for driving in urban environments. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, pp. 237–252, 2018. URL http://proceedings.mlr.press/v87/sauer18a.html.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

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

Sergeev, A. and Del Balso, M. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065, 2017. URL http://arxiv.org/abs/1705.05065.

Sünderhauf, N., Brock, O., Scheirer, W. J., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., and Corke, P. The limits and potentials of deep learning for robotics. *CoRR*, abs/1804.06557, 2018. URL http://arxiv.org/abs/1804.06557.

Tan, B., Xu, N., and Kong, B. Autonomous driving in reality with reinforcement learning and image translation. *CoRR*, abs/1801.05299, 2018. URL http://arxiv.org/abs/1801.05299.

Tobin, J., Biewald, L., Duan, R., Andrychowicz, M., Handa, A., Kumar, V., McGrew, B., Ray, A., Schneider, J., Welinder, P., Zaremba, W., and Abbeel, P. Domain randomization and generative models for robotic grasping. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pp. 3482–3489, 2018. doi: 10.1109/IROS.2018.8593933. URL https://doi.org/10.1109/IROS.2018.8593933.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. URL https://doi.org/10.1109/IROS.2012.6386109.