Supplementary Material for: Towards Safe Reinforcement Learning via OOD Dynamics Detection in Autonomous Driving System.

Arnaud Gardille, [1] **Ola Ahmad,** [2]
[1] Paris-Saclay University
[2] Thales Digital Solutions, Montreal, Canada
arnaud.gardille@universite-paris-saclay.fr ola.ahmad@thalesdigital.io

Implementation details

We provide in this supplemental some details of our implementation.

Environment

The use of CARLA as a gym environment is illustrated on Figure 1. We used the docker-contained version 0.9.6 of the simulator through a gym embedding available at https://www.github.com/cjy1992/gym-carla. We adapted it to the needs of our experiments though gym wrappers. Our code is implemented in Python under Ubuntu 20.04. The agent only chooses between breaking and accelerating. The steering is done automatically, and the agent only perceives the speed and distance to the obstacle. All training experiments take place in one map (Town01).
Formally, the action space is $[-5, 5]$, with positive values corresponding to acceleration and negative values to braking. The observation space is $[0, 32] \times [0, 20]$, where the first component is the obstacle's distance in meters, and the second corresponds to the car's speed in m/s. If the perceived obstacle's distance is far, its value is then truncated at 32 meters.

The reward function is

$$r = 200.0 * r_{collision} + 1.0 * r_{speed} + 10.0 * r_{proximity}$$

with

$$r_{speed} = \begin{cases} v_{desired} - (v_{real} - v_{desired})^2 & \text{for } d < 12 \\ -v_{real} & \text{otherwise} \end{cases}$$

$$r_{collision} = -(v_{real} + 1.0) \quad \text{in case of collision}$$

$$r_{proximity} = \begin{cases} d_{hazard} - 5 & \text{for } d_{hazard} < 5 \\ 0 & \text{otherwise} \end{cases}$$

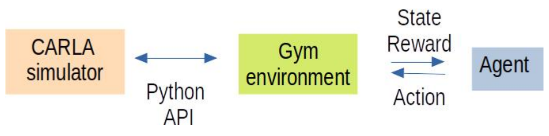

Figure 1: Method's architecture

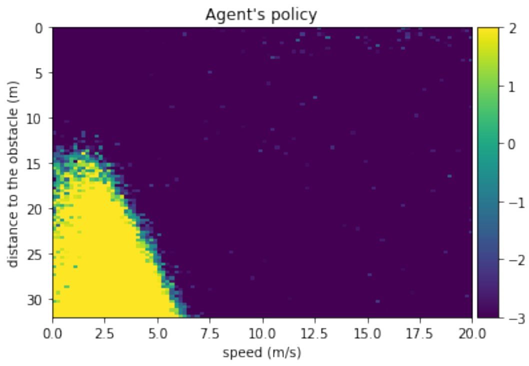

Figure 2: Policy of the trained SAC agent, with the position corresponding to the state, and the colour to the action given in response. We can see the agent accelerates when it is slow and far from the obstacle, and breaks otherwise.

Agent

We used the Stable-Baselines3 library to train a Soft Actor Critic (SAC) agent over 3 millions steps. This algorithm was chosen because we use continuous state and action spaces, an privilege sample efficiency over speed because we cannot run several instances of the environment in paralleled. The agent starts on randomly selected valid point on on the map, and the obstacle was randomly selected among the available bikers and vehicles. Figure 2 represents the policy of our SAC agent.

Feature Extractor

We trained a feature extractor to predict the distance to the front obstacle from the front RGB camera. The input image shape is 224*224, in order to make use a version of the AlexNet network that was pre-trained on ImageNet. We fine-tuned it replacing the last layers by a fully connected one leading to a single neuron without activation function. We trained it using the Adam optimizer during 25 epochs on a dataset of 10000 examples. It was trained only on the ClearNoon weather of the CARLA simulator, in order to face visual OOD when used on the HardRainSunset weather. You can observe the evolution of the relative error

$$\frac{d_{estimated} - d_{real}}{d_{real}}$$

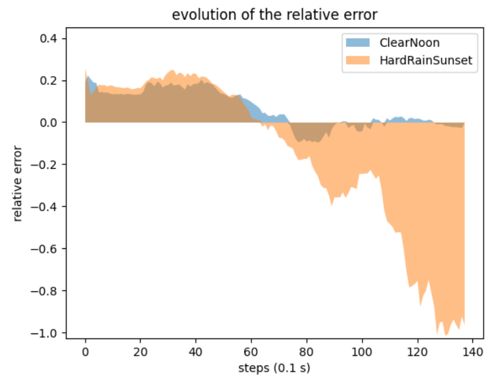

Figure 3: Evolution of the feature extractor's relative error through an episode. The blue graph correspond to the original weather, and the orange graph to the OOD weather. In this context, the change of weather led the feature extractor to greatly overestimate the distance when it is near the obstacle. This caused any of our efficient agents to collide.

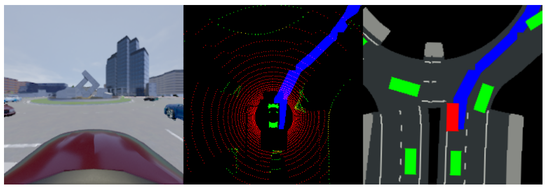

Figure 4: Visual observations available in the original gym environment. The front rgb image is on the left, the LIDAR image in the cented and the generated bird view of the scene is on the right.

through an episode on Figure 3. The use of the lidar information as presented on Figure 4 leads to a better estimation of the distance. We didn't used it because it would have made the OOD analysis too complicated.

Experiments Configuration

In order to make our experiments reproducible, and have a fair comparison between the standard and OOD situations, we fixed the spawning point of the agent's vehicle and spawning distance of the obstacle.

Model

The environment's model takes as input the previous actions and observations of the 50 previous steps. A first network predicts the state difference between the previous and the current one. It is then added to the current state to get an approximation of the current state. Predicting the states dynamic proved to be much more efficient than to directly predict the next state. The second network predicts $g(S_{t-1}, A_{t-1}) \approx \sigma_t$, the standard deviation of the model's prediction deviation.
Those fully connected networks share the same architecture: The size of the hidden layers are [1000, 500, 200, 100]. It uses the ReLu activation function, a dropout of 0.1 as a regulariser, and a learning rate decreasing from 1e-4 to 1e-7 through the 25 epochs of training. Those networks are trained with the L_2 norm for regression.
The impact of approximate normalization can be seen in figure 6. After dividing the prediction deviation by the estimated standard deviation, we observe that the flat purple area at the top becomes much more expressive at the bottom.

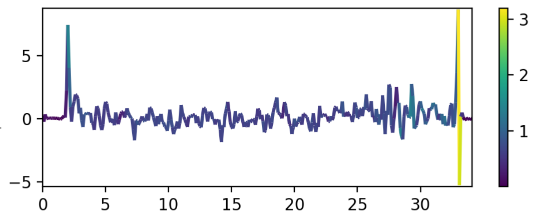

Figure 5: Normalized model's gap predicting speed's difference through an episode, coloured by locally estimated standard deviation $g(S_{t-1}, A_{t-1}) \approx \sigma_t$. The x-axis is in second.

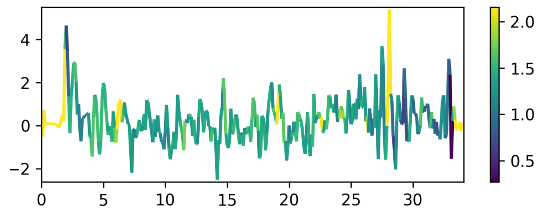

Figure 6: Model's gap predicting speed's difference through an episode, approximately normalized locally by dividing it by the estimation $g(S_{t-1}, A_{t-1}) \approx \sigma_t$. the graph is coloured according to the multiplicative factor due to normalisation. The x-axis is in second.