RL project submission template

Carraro Eddie 2121248

Abstract

This report presents the results of applying different reinforcement learning approaches to an autonomous driving environment, where the agent must make real-time decisions for an ego-vehicle in order to maximize its cumulative reward (for example, by avoiding collisions with other vehicles). The results are compared across the different reinforcement learning methods, as well as against a manual control mode and a baseline approach implemented programmatically without Reinforcement Learning.

The experiments show that Deep Q-Network (DQN) methods outperform Proximal Policy Optimization (PPO) and Vanilla Policy Gradient (VPG). Specifically, PPO and VPG tend to converge to a local optimum, whereas Dueling-DQN emerges as the most effective approach in this setting, closely followed by Double-DQN.

1. Introduction

The goal of this task is to enable an autonomous vehicle to safely and efficiently drive through a highway populated with other vehicles, by means of a Reinforcement Learning (RL) agent. The documentation of the environment used for training and evaluation can be found at https://highway-env.farama.org/ .

Since the current state of the art in autonomous driving research largely relies on Deep Reinforcement Learning (DRL) algorithms, this project adopts the same approach. However, the choice of the most suitable algorithm for this specific task was not immediately clear, so multiple algorithms were tested and compared.

Selecting the right DRL method for a given task is crucial, as different algorithms exhibit varying strengths and limitations. For instance, the Proximal Policy Optimization (PPO) algorithm is generally considered reliable, as it tends to avoid risky actions by favoring safer strategies, even if these yield lower rewards. Nevertheless, other algorithms may outperform PPO by adopting more effective decision-making strategies, even if less safe than PPO.

2. Critical Issues of the Project

At first glance, the highway environment may seem straightforward: the ego-vehicle simply needs to drive fast while avoiding collisions. However, the environment actually poses several challenges. The agent must continuously track the positions, velocities, and behaviors of surrounding vehicles, which change rapidly and unpredictably. This makes decision-making difficult and increases the complexity of the learning problem.

In particular, we will observe that PPO struggles with this complexity, while VPG converges to a simple but suboptimal strategy that avoids crashes without achieving high returns.

3. Manual Control

Using the Manual Control mode, it is possible to experiment with the environment and observe the rewards associated with different actions of the ego-vehicle. In particular, for each episode, it is possible to achieve a reward close to 39.5/40 by maintaining the rightmost lane and driving at full speed, while avoiding collisions with other vehicles.

4. My Baseline

The baseline policy was implemented programmatically, without using Reinforcement Learning, but instead through a set of handcrafted heuristics derived from manual control experiments. Its goal is to provide a simple, interpretable, yet competitive reference point against which RL agents can be compared.

4.1. Design Principles

The baseline was designed around the following rules:

- **Rightmost lane preference:** The agent always attempts to move to the rightmost lane whenever possible, as this is explicitly rewarded in the environment.
- **Speed adaptation:** If no vehicles are detected within a safe longitudinal distance ahead, the agent accelerates. If vehicles are close ahead, the agent decelerates to avoid collisions.

- **Lane changes:** If a vehicle is detected directly in front of the ego-vehicle:
  - Prefer changing to the right lane, if it is clear.
  - Otherwise, change to the left lane if it is clear.
  - If both adjacent lanes are blocked, slow down.
- **Dynamic safety distance:** The minimum safe following distance increases proportionally to the ego-vehicle's velocity (e.g., larger at higher speeds).

4.2. Algorithm

The policy can be summarized as follows:

1. At the start of each episode, reduce speed and attempt to reach the rightmost lane within the first few steps.
2. At each subsequent timestep:
   (a) If no vehicle is detected within the safe distance ahead, select the `FASTER` action.
   (b) If a vehicle is detected in front:
   - If the right lane is free, select `LANE_RIGHT`.
   - Else if the left lane is free, select `LANE_LEFT`.
   - Else, select `SLOWER`.

4.3. Performance

Over 10 evaluation episodes, this baseline achieves an average return of approximately 29 ± 0.5 (see Fig. 1). This makes it significantly stronger than a trivial baseline (e.g., driving slowly in a straight line) because it is capable of overtaking and lane selection. However, the baseline is still quite rigid: it always follows the same fixed rules and cannot really adapt when the traffic situation becomes more complex or when it would need to balance speed and safety in smarter ways.

This baseline is reasonable because:

- It captures essential safe-driving heuristics (lane discipline, adaptive speed).
- It is interpretable and simple to reproduce.
- It achieves non-trivial returns, making it a meaningful benchmark for evaluating whether RL agents truly learn better driving behaviors.

5. Vanilla Policy Gradient (VPG) Algorithm

Before implementing PPO, I first developed a Vanilla Policy Gradient (VPG) implementation, available in **implementation_vpg.py**, with the corresponding training script in **training_vpg.py**. This method is relatively simple to implement, but its performance quickly converges to a local optimum with an average return of around 30. In particular, the agent learns to drive more slowly to avoid crashes, rather than attempting risky overtakes or higher speeds that could result in collisions. However, the agent does not learn to remain in the rightmost lane. The rewards obtained during training are reported in Fig. 2.

6. Proximal Policy Optimization (PPO) Algorithm

The implementation of the PPO algorithm (Schulman et al., 2017) is provided in **implementation_ppo_clip.py**, with the training procedure defined in **training_ppo.py**. To the standard implementation, I added several improvements, including *learning rate annealing*, *entropy regularization*, *gradient clipping* (to stabilize training when gradients become too large), and an *approximation of KL divergence* to monitor the difference between the new and old policy distributions during learning.

Since PPO is generally more conservative, it typically requires a larger number of training episodes to converge. For this reason, I extended the training duration to better evaluate its performance. However, as noted earlier, the results obtained with PPO were not particularly strong, suggesting that this algorithm may not be well-suited to this type of environment.

During training, the loss values were also recorded. Nevertheless, in the case of PPO, these values are not highly informative, as they tend to fluctuate significantly even when learning is proceeding effectively. The results can be seen in Fig. 3, 4, 5, and 6.

7. DQN Algorithm

The implementation of the DQN algorithms is provided in **implementations_dqn.py**, with the corresponding training procedure in **training_dqn.py**. The performance obtained with DQN-based approaches was superior to that of PPO. In particular, two variants were evaluated: Double-DQN and Dueling-DQN (trainable from the same file, deciding if to train Double-DQN or Dueling-DQN from command line). The latter demonstrated slightly better performance, although both approaches achieved strong and consistent results overall.

7.1. Double-DQN

The episode returns and losses I got using Double-DQN algorithm (Van Hasselt et al., 2016) can be seen in Fig. 7, 8 and 9. Performances are higher with respect to our baseline.

7.2. Dueling-DQN

The episode returns and losses I got using Dueling-DQN algorithm (Wang et al., 2016) can be seen in Fig. 10, 11 and 12. Also in this case, performances are higher with respect to the baseline I made.

8. Hyperparameters

8.1. VPG Used Hyperparameters

Table 1. VPG hyperparameters

Hyperparameters	
Training episodes	2×10^4
Learning rate	10^{-3}
Number of hidden features of networks	256

8.2. PPO Used Hyperparameters

Table 2. PPO hyperparameters

Hyperparameters	
Training episodes	4×10^4
Discount factor	0.99
Batch size	4000
Actor learning rate	3×10^{-4}
Critic learning rate	3×10^{-4}
Number of hidden features of networks	128
PPO steps	4
Epsilon	0.2
Entropy coefficient	0.01
Clip threshold	0.2
Max grad norm	0.5
Target KL divergence value	0.02

8.3. DQN Used Hyperparameters

Table 3. DQN hyperparameters

Hyperparameters	
Training episodes	3×10^4
Discount factor	0.99
Buffer size	10000
Batch size	128
Learning rate	0.001
Number of hidden features of networks	256
Epsilon	1.0
Epsilon decay rate	0.995
Epsilon (min value)	0.01
Target update frequency	1000

9. Figures

9.1. My Baseline Results

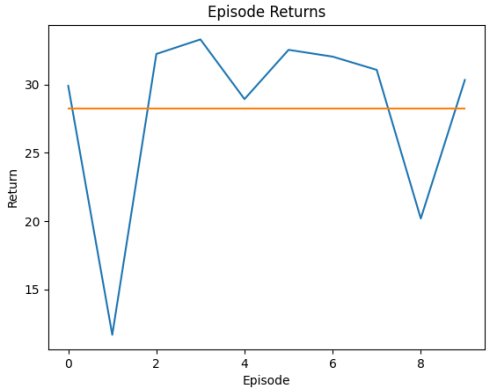

Figure 1. Baseline returns over 10 episodes. The red line indicates the mean value.

9.2. VPG Results

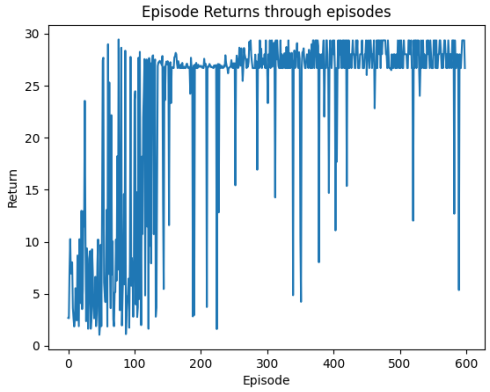

Figure 2. VPG total returns during 600+ episodes of training.

9.3. PPO Results

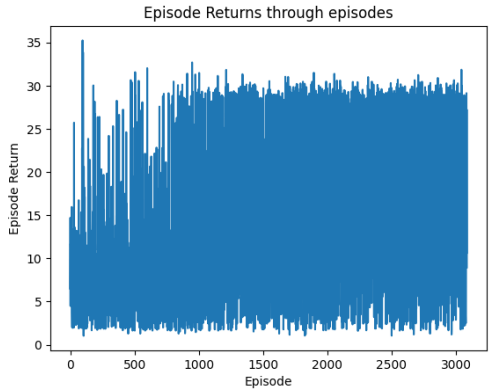

Figure 3. PPO total returns during 2000+ episodes of training.

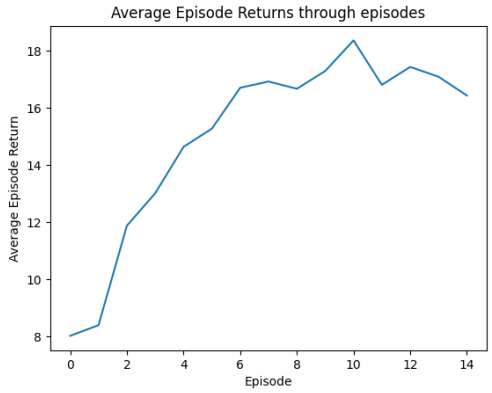

Figure 4. PPO mean returns computed considering each rollout during 2000+ episodes of training.

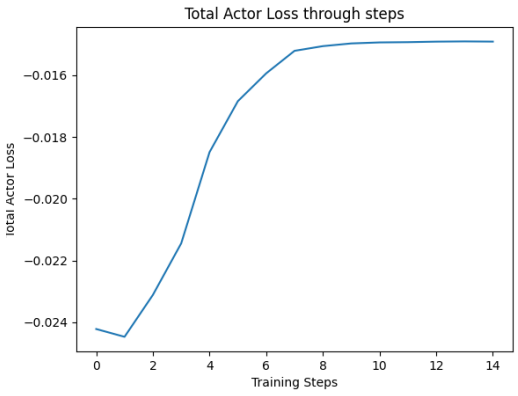

Figure 5. PPO total actor losses during 2000+ episodes of training.

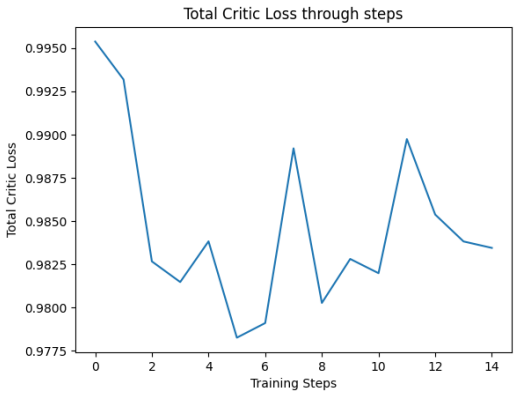

Figure 6. PPO total critic losses during 2000+ episodes of training.

9.4. Double-DQN Results

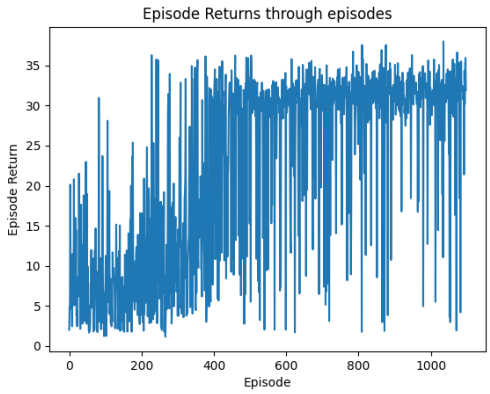

Figure 7. Double-DQN total returns during 1000+ episodes of training.

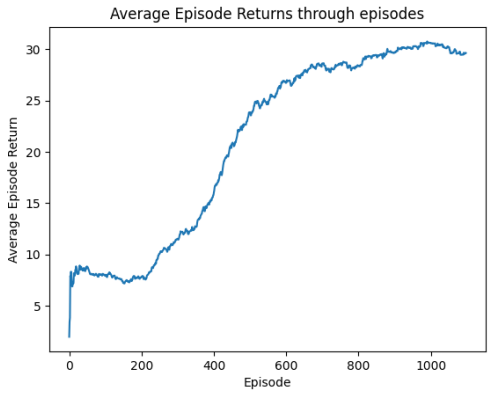

Figure 8. Double-DQN mean returns computed considering the last 100 episodes during 1000+ episodes of training.

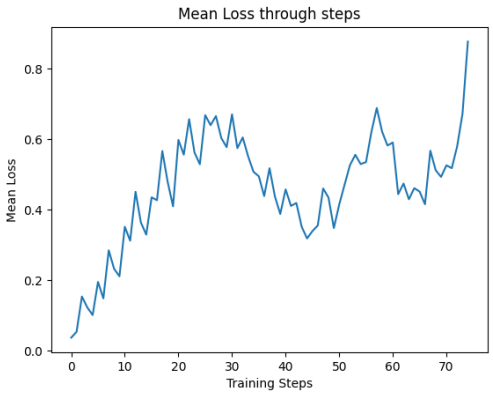

Figure 9. Double-DQN mean losses computed considering the last 100 episodes during 1000+ episodes of training.

9.5. Dueling-DQN Results

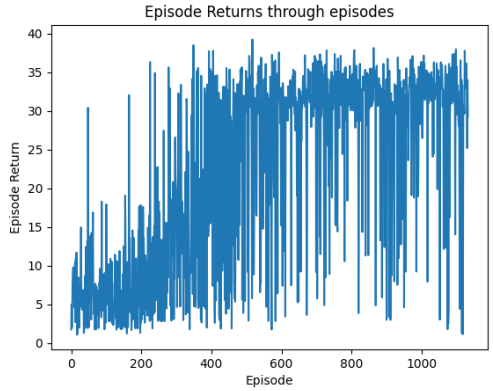

Figure 10. Dueling-DQN total returns during 1000+ episodes of training.

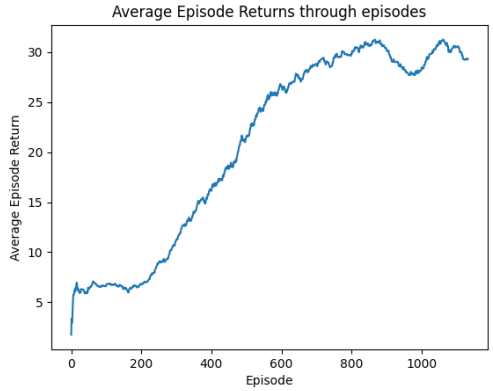

Figure 11. Dueling-DQN mean returns computed considering the last 100 episodes during 1000+ episodes of training.

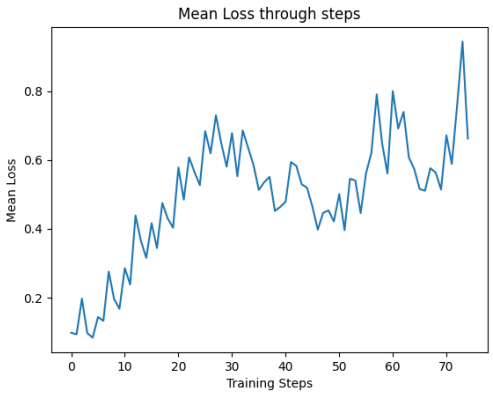

Figure 12. Dueling-DQN mean losses computed considering the last 100 episodes during 1000+ episodes of training.

9.6. Dueling-DQN evaluation

An example of an evaluation using the Dueling-DQN algorithm, which achieved the best performance among all tested methods, is also presented here.

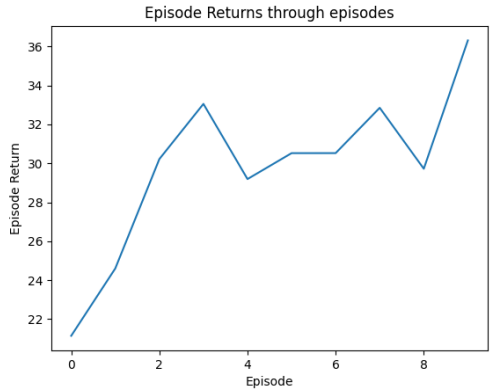

Figure 13. Dueling-DQN evaluation example.

10. Comments

From the plots, it is evident that VPG exhibits generally more stable behavior compared to PPO, maintaining returns around 30 over many episodes, whereas PPO's returns are highly variable. This stability arises because VPG, as previously mentioned, keeps lowering the ego-vehicle speed (until a minimum value) and avoids risky actions, without exploring alternative strategies to improve future returns. Consequently, VPG converges to a local optimum: its performance is more stable but slightly lower than that of PPO, which explores more aggressively and exhibits greater variability in returns.

11. Conclusions

Among the tested methods, VPG proved to be the simplest to implement. Its simplicity, however, also makes it prone to converging to a local optimum. In this case, the learned strategy involves keeping the ego-vehicle at a lower speed to avoid collisions, without exploiting the possibility of maintaining higher speeds or consistently remaining in the rightmost lane. While VPG may perform adequately in simpler environments, its limitations become clear in more complex driving scenarios.

PPO remains a widely used and well-regarded algorithm in Deep Reinforcement Learning due to its robustness and relative ease of implementation. Nonetheless, its shortcomings are evident in this work: its overall performance is limited, likely influenced by the nature of the environment, and it is outperformed by other methods such as Deep Q-Networks.

In contrast, the DQN-based algorithms demonstrated performance much closer to that of manual control, and significantly better than the programmatically defined baseline developed for this project. Among them, the Dueling-DQN emerged as the most effective approach, offering strong performance and more consistent decision-making compared to the other tested methods.

References

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

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.