CentraleSupélec

Report

Mention IA

RL class Project

Reinforcement Learning for Autonomous Driving

Students:
Samuel Sithakoul
samuel.sithakoul@student-cs.fr
Paul Massey
paul.massey@student-cs.fr

Supervisors:
Haji Hediji

April 24, 2025

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful framework for developing autonomous agents capable of making complex decisions in dynamic environments. One particularly relevant application is autonomous driving, where agents must learn to navigate safely and efficiently through a variety of road scenarios. In this report, we investigate the training and evaluation of RL-based car agents using the highway-env simulation framework, which provides diverse and customizable driving environments.

Our study focuses on three distinct environments within highway-env: Highway, Roundabout, and Racetrack. Each presents unique challenges that test different aspects of the agent's learning and decision-making capabilities. By training agents in these environments, we aim to assess how well RL methods generalize across different driving tasks and road structures. We explore the performance of various algorithms, analyze their learned behaviors, and highlight the strengths and limitations of current approaches in handling diverse driving scenarios.

This part of the study done here covers these 3 environments with a Deep-Q Learning algorithm, Soft Actor Critic (SAC) and a Proximal Policy Optimization (PPO). Other experiments are done in Paul Massey's work with a Deep Deterministic Gradient instead of SAC and a DQN instead of PPO with the addition of testing more behaviours on the environments.

Our code is available publicly at this repository `https://github.com/Rubiksman78/BatmoobileRL`.

2 Task 1: Highway

2.1 Task description

In the Highway environment, the agent vehicle is driving on a multi-lane highway with other vehicles predetermined and driving at the same time. The agent's objective is to reach a high speed while avoiding collisions with neighbouring vehicles and also with a reward of driving on the right lane (because we are not in the United States).

This environment has a default configuration but we will use another one as defined in the problem statement:

As opposed to the default configuration, we will be using the OccupancyGrid observation which gives features in a grid around the agent instead of Kinematics which directly gives features of nearby vehicles. The right lane reward is also more important here than the high speed reward which might be closer to a real case where we want to put more importance on respecting driving rules.

2.2 Deep Q Learning

We implement the Deep Q Learning algorithm such as seen in class and practicals. No specific trick is used except from exponential decay of ϵ for exploration which is already common practiced with DQN.

2.3 Training

We implement the previously introduced algorithm from scratch in PyTorch with the following set of hyperparameters:

Parameter	Default config	Used config
observation.type	Kinematics	OccupancyGrid
observation.vehicles_count	-	10
observation.features	-	presence, x, y, vx, vy, cos_h, sin_h
observation.features_range	-	x:[-100,100], y:[-100,100]vx:[-20,20], vy:[-20,20]
observation.grid_size	-	[[-20, 20], [-20, 20]]
observation.grid_step	-	[5, 5]
observation.absolute	-	False
action.type	DiscreteMetaAction	DiscreteMetaAction
lanes_count	4	4
vehicles_count	50	15
duration	40 [s]	60 [s]
initial_spacing	2	0
collision_reward	-1	-1
right_lane_reward	0.1	0.5
high_speed_reward	0.4	0.1
lane_change_reward	0	0
reward_speed_range	[20, 30] [m/s]	[20, 30] [m/s]
simulation_frequency	15 [Hz]	5 [Hz]
policy_frequency	1 [Hz]	1 [Hz]
screen_width	600 [px]	600 [px]
screen_height	150 [px]	150 [px]
centering_position	[0.3, 0.5]	[0.3, 0.5]
scaling	5.5	5.5
show_trajectories	False	False
render_agent	True	True
offscreen_rendering	False	False
disable_collision_checks	-	True

Table 1: Comparison of the two environment configurations

Parameter	Symbol	Value
Total timesteps	T_{total}	2×10^5
Batch size	B	64
Discount factor	γ	0.99
ϵ start	ϵ_{start}	1.0
ϵ end	ϵ_{end}	0.02
ϵ decay	τ_ϵ	$0.1T_{\text{total}}$
Target update rate	τ	0.005
Learning rate	α	2×10^{-4}
Warmup steps	t_{warmup}	200
Update interval	f_{target}	50
Replay buffer size	$\lvert\mathcal{B}\rvert$	2×10^5
Hidden dimension	N_{hidden}	64

Table 2: DQN Hyperparameters

2.4 Results

To check the results of our algorithm in practice, we compute the total reward obtained over an episode for 1000 simulations and we plot the distribution of the obtained rewards.

We can see here that we obtain consistently a reward close to the maximum reward which is around 60 because of a duration of 60s in the environment. Some rewards are in the lower part because we can suppose there are some failure cases with crashes. Rewards don't all reach 60 for example if the car is starting at the upmost left lane so the reward for right lane positioning is not gotten from the first step right away. To enforce a more active behaviour where the car overtakes other cars to go faster, a more important coefficient could be given to the high speed reward.

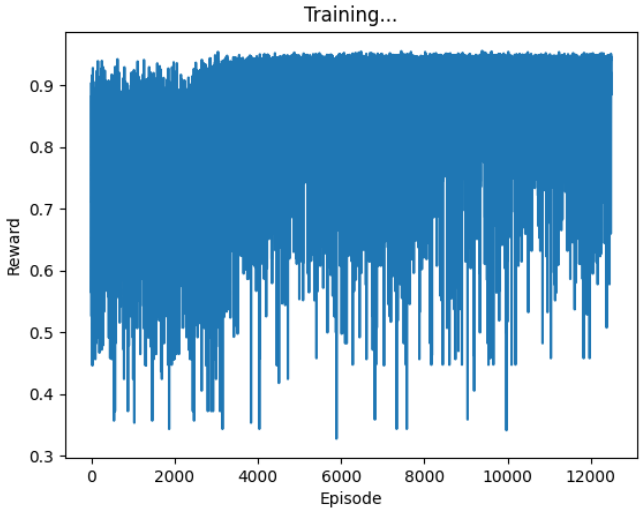

Figure 1: DQN reward evolution during training

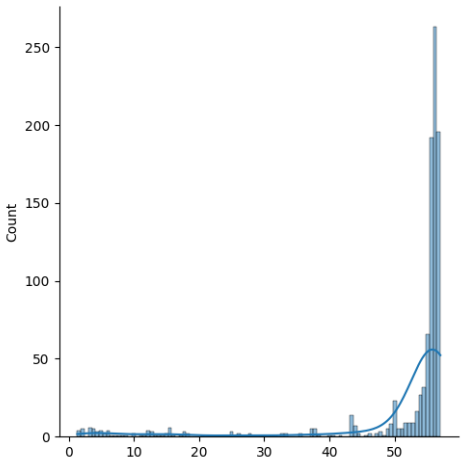

Figure 2: DQN reward distribution on 1000 simulations

Figure 3: The car is going to the right and stays here until the end

3 Task 2: Racetrack

3.1 Task description

Racetrack is a continuous control environment where the agent vehicle has to follow the tracks of a looping circuit. The agent's objective is to avoid collisions with neighbouring vehicles while staying centered on the track.

The configuration used is the default one:

Parameter	Value
observation.type	OccupancyGrid
observation.features	presence, on_road
observation.grid_size	[[-18, 18], [-18, 18]]
observation.grid_step	[3, 3]
observation.as_image	False
observation.align_to_vehicle_axes	True
action.type	ContinuousAction
action.longitudinal	False
action.lateral	True
simulation_frequency	15 [Hz]
policy_frequency	5 [Hz]
duration	300 [s]
collision_reward	-1
lane_centering_cost	4
action_reward	-0.3
controlled_vehicles	1
other_vehicles	1
screen_width	600 [px]
screen_height	600 [px]
centering_position	[0.5, 0.5]
scaling	7
show_trajectories	False
render_agent	True
offscreen_rendering	False

Table 3: Configuration for Racetrack

There is also a *action_reward* which penalizes when the agent makes too many corrections leading to unstable driving.

3.2 Soft Actor Critic

More details about the algorithm can be found in the Appendices 7. We use SAC for better stability and exploration, especially with the choice of the learnt entropy regularization implementation.

3.3 Training

We implement the previously introduced algorithm with the defined loss equations from scratch in PyTorch. For simplicity, we don't use vectorized environment and keep a simple loop over sequential episode rollout. The choice of fixed entropy weight or automatic learning of the entropy regularization is implemented but we mainly use the automatic version for better convergence.

The agent converged rapidly closed to the maximum possible rewards thanks to the exploration and stability of SAC. As we can see on the average episode length, at the beginning the agent has a lot of crashes which causes the episode to terminate before the maximum duration while at the end this duration is reached consistently.

3.4 Results

The simulation for the race track environment is longer than other environments (approximately 300s) so we accumulate the rewards here for 100 simulations.

Parameter	Symbol	Value
Discount factor	γ	0.99
Target update rate	τ	0.005
Replay buffer size	$\lvert\mathcal{B}\rvert$	10^6
Batch size	B	128
Learning starts steps	t_{init}	5000
Max episode steps	$T_{\max}$	5000
Total episodes	N_{episodes}	5000
Policy update frequency	f_π	2
Policy learning rate	α_π	3×10^{-4}
Critic learning rate	α_Q	3×10^{-4}
Entropy coefficient	α	0.2 (auto)
Min log std	$\log \sigma_{\min}$	-5
Max log std	$\log \sigma_{\max}$	2
Hidden dimension	N_{hidden}	256

Table 4: SAC Hyperparameters

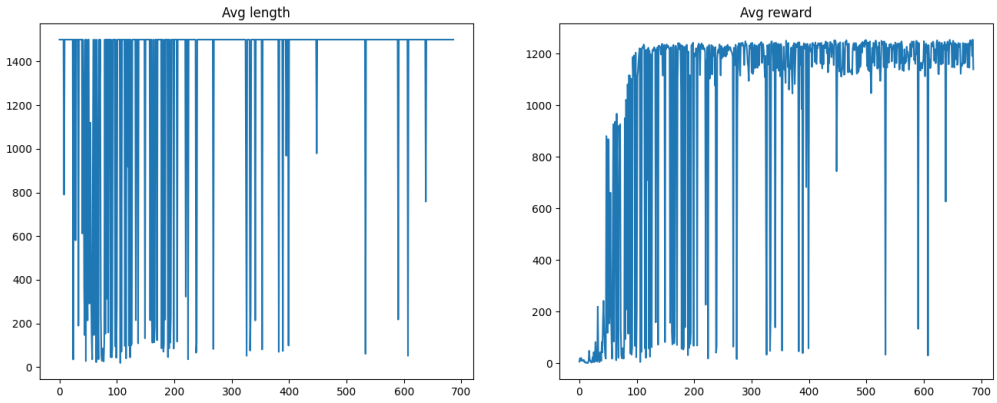

Figure 4: Evolution of total rewards obtained during an episode and episode length during training

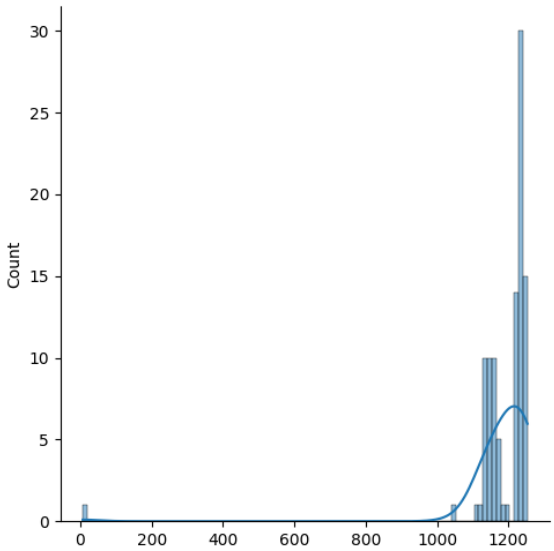

Figure 5: SAC reward distribution on 100 simulations

The SAC agent here obtains a reward higher than 1000 consistently which shows that it can stay on track during the whole duration of the episode while avoiding collisions with the other vehicle.

When checking on rendered examples, we see that the agent follows the track with small adjustments at each step without crashing into the other car. However, it seems that the agent hacked the

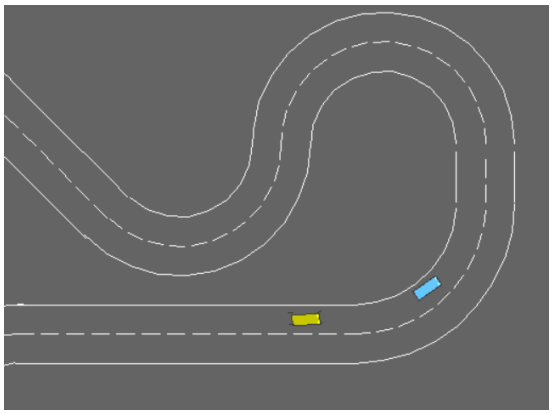

Figure 6: The car follow the track correctly with some shaking

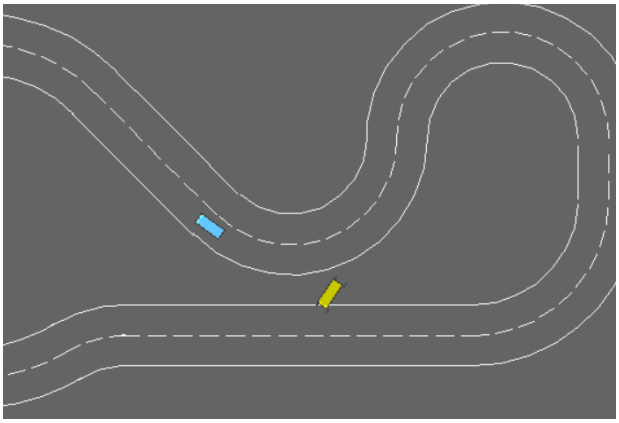

Figure 7: When overtaking, the car tends to change line to avoid the issue

reward to avoid collision when getting close to the other car, it learnt to go in the opposite direction by crossing the track so it doesn't have to worry about this issue in the future. Given the long duration of the episode, the reward lost for leaving the track is not significant and thus explains this behaviour. To fix this issue, rewards penalizing the switch of direction could be implemented.

4 Task 3: Roundabout

4.1 Task description

In this case, the agent vehicle is approaching a roundabout with flowing traffic. It follows its planned route automatically but has to handle lane changes and longitudinal control to pass the roundabout as fast as possible while avoiding collisions.

We modified the initial configuration to have a better representation of a real case scenario and based on suggestions from the library author.

Parameter	Config 1 (Kinematics)	Config 2 (GrayscaleObservation)
`observation.type`	Kinematics	GrayscaleObservation
`observation.absolute`	True	-
`observation.observation_shape`	-	(128, 128)
`observation.stack_size`	-	4
`observation.weights`	-	[0.2989, 0.5870, 0.1140]
`observation.scaling`	-	1.75
`observation.features_range`	`x:[-100,100], y:[-100,100]` `vx:[-15,15], vy:[-15,15]`	-
`action.type`	DiscreteMetaAction	DiscreteMetaAction
`action.target_speeds`	[0, 8, 16]	-
`incoming_vehicle_destination`	None	None
`collision_reward`	-1	-4
`high_speed_reward`	0.2	0.2
`right_lane_reward`	0	0
`lane_change_reward`	-0.05	-0.05
`duration`	11 [s]	11 [s]
`simulation_frequency`	-	15 [Hz]
`policy_frequency`	-	1 [Hz]
`screen_width`	600 [px]	600 [px]
`screen_height`	600 [px]	600 [px]
`centering_position`	[0.5, 0.6]	[0.5, 0.6]
`scaling`	-	5.5
`normalize_reward`	True	True

Table 5: Comparison Between Kinematics and GrayscaleObservation Configurations

In our modified configuration, we use image observations in grayscale with stacking of four consecutive frames as done in many classical algorithms with image inputs. This choice was encouraged by

issues encountered with the development of RL algorithms in this environment and recommendations from the author in this issue `https://github.com/Farama-Foundation/HighwayEnv/issues/99`. The motivation is that Kinematics or OccupancyGrid contains ordered information about the vehicles which is not preferred with Fully connected Neural Networks which are not invariant by permutations of the inputs. However the combination of image input and CNN might be better as it is not dependant on the order of vehicles and better associated with the notion of a field of view for the agent. We also put more weight to the penalty for collision as the model might tend to prioritize its speed which gives an overall higher total reward across an episode.

4.2 Proximal Policy Optimization

More details about the algorithm can be found in the Appendices 7. Some classical improvements are also used such as vectorized environments, General Advantage Estimation and entropy regularization.

4.3 Training

For the implementation with StableBaselines3, we use the following hyperparameters and let the others with their default values:

Parameter	Symbol	Value
Environments	N_{envs}	8
Batch size	B	128
Rollout steps	T	2048
GAE parameter	λ	0.95
Discount factor	γ	0.99
Clip range	ϵ	0.2
Training epochs	K	6
Entropy coefficient	α	0.001
Total timesteps	T_{total}	5×10^5

Table 6: PPO Hyperparameters

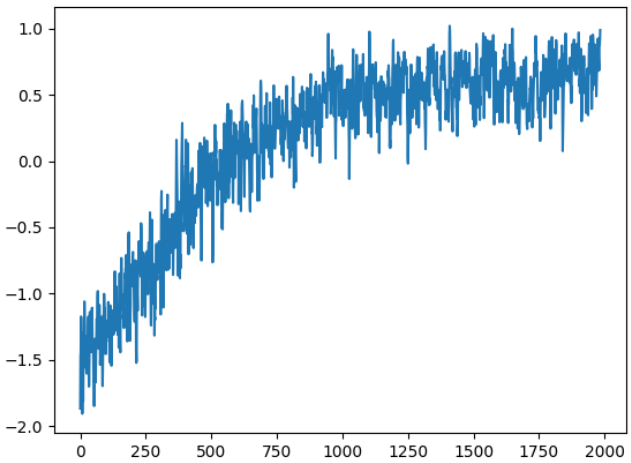

Figure 8: PPO reward across training, moving average is taken over 100 steps for better readability

The moving average reward evolves during training towards 1 which indicates that there are fewer crashes in general and more rewards obtained for high speed.

4.4 Results

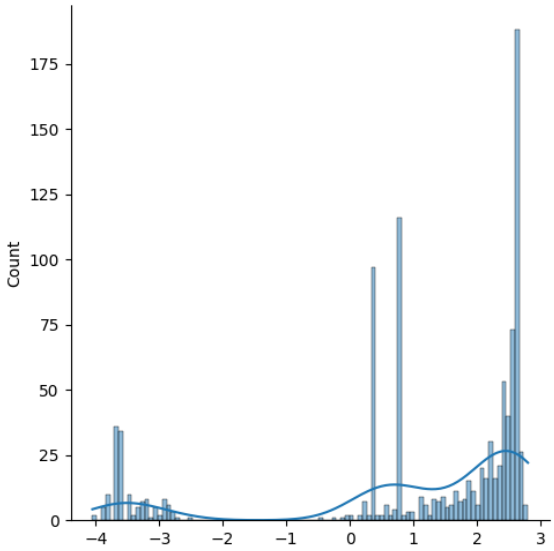

Figure 9: PPO reward distribution on 1000 simulations

The behaviour observed during training is confirmed in inference with many rewards closed to a very high value which indicates that the car crosses safely the roundabout at a high speed. A part of the simulation give a rewards between -4 and -3 which represents a collision at the start of the roundabout.

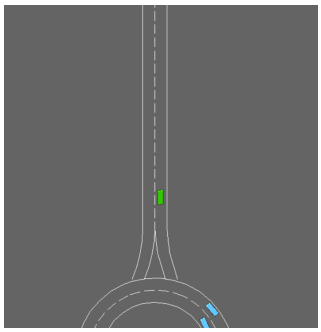

Figure 10: The car exits the roundabout safely and quickly

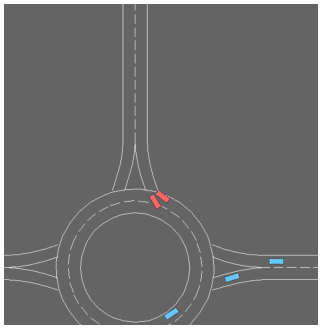

Figure 11: Another car cuts from the left leading in a crash

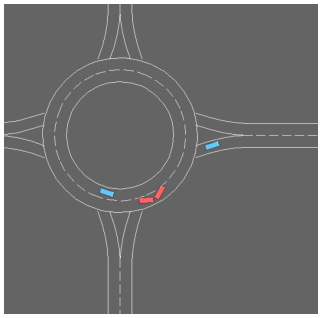

Figure 12: The agent car enters while another car is coming

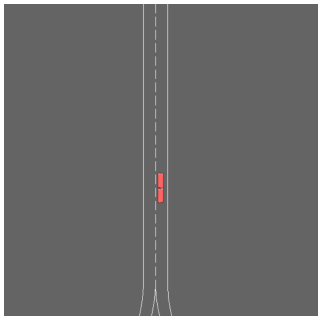

Figure 13: The agent crashes into another car after exiting

The global results are also confirmed in rendered simulations. The car often crosses the roundabout but due to the highly random behaviour of the environment, there are collisions unavoidable when another car cuts the lane. These cases encourage the agent to still cross at a high speed because in general this behaviour works when there are no car arriving at the same time. To encourage a safer behaviour, the penalty for collision could be increased but this has the risk of leading to an unwanted behaviour of the car not entering the roundabout at all. This instability of the environment for learnt agents is discussed by the author of the library at this issue https://github.com/Farama-Foundation/HighwayEnv/issues/99.

5 Task 4: Extra experiment

This is handled by my teammate Paul Massey. Different approaches are possible here, for example testing trained agents on other environments with a similar goal (highway and roundabout to some extent).

6 Conclusion

This work explored three different environments for autonomous driving with different goals and settings. In some easier environment, trained agents converged easily but such convergence is harder to obtain in more complex environments where there are a lot of random behaviours involved such as the roundabout case. Agents are also very dependant on the reward given and can tend to do *reward hacking* when it is not aligned with the true objective desired. Further work could explore situations closer to real-life situations, for example with a camera view from the driver or try other algorithms to solve the harder cases of highway-env.

7 Appendices

Task 2 : Soft-Actor-Critic

Soft Actor-Critic (SAC) is an off-policy reinforcement learning algorithm based on the maximum entropy framework, which optimizes policies to maximize both expected reward and entropy. This encourages exploration and robustness, making it particularly effective in continuous control tasks.

SAC consists of:

- A stochastic policy (actor) $\pi_\phi(a|s)$ with parameters ϕ.
- Two Q-functions (critics) $Q_{\theta_1}(s,a)$ and $Q_{\theta_2}(s,a)$ to mitigate overestimation bias.

The policy is trained to maximize:

$$\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t)+\alpha\mathcal{H}(\pi(\cdot|s_t))\right)\right], \tag{1}$$

where α is the temperature parameter controlling entropy regularization, and the entropy is defined by

$$\mathcal{H}(\pi(\cdot|s)) = \mathbb{E}_{a\sim\pi}[-\log\pi(a|s)] \tag{2}$$

The Q-functions are updated by minimizing the Bellman error:

$$\mathcal{L}_Q(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_{\theta_i}(s,a)-\left(r+\gamma\left(\min_{j=1,2}Q_{\bar{\theta}_j}(s',a')-\alpha\log\pi_\phi(a'|s')\right)\right)\right)^2\right], \tag{3}$$

where $\mathcal{D}$ is the replay buffer and $\bar{\theta}_j$ are target network parameters. Similarly to Twin Delayed DDPG (TD3), the minimum of two Q-values is taken to improve stability.

The policy parameters ϕ are updated to maximize:

$$\mathcal{L}_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\phi}\left[\alpha\log\pi_\phi(a|s)-\min_{j=1,2}Q_{\theta_j}(s,a)\right], \tag{4}$$

using the reparameterization trick for gradient estimation.

The temperature α is optimized to maintain a target entropy $\bar{\mathcal{H}}$:

$$\mathcal{L}(\alpha) = \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\phi}\left[-\alpha\left(\log\pi_\phi(a|s)+\bar{\mathcal{H}}\right)\right]. \tag{5}$$

It is also possible as done in the early versions of the algorithm to use a constant temperature but we prefer this learned version to improve exploration.

Task 3: PPO

We use Proximal Policy Optimization as seen in class with the implementation from StableBaselines3 for discrete actions. PPO is adaptable to discrete control tasks by modifying the policy representation and advantage estimation while retaining its clipped objective.

In discrete control, the policy $\pi_\theta(a|s)$ is typically a categorical distribution over actions, parameterized by a neural network with a softmax output layer:

$$\pi_\theta(a|s) = \frac{e^{f_\theta(s)_a}}{\sum_{a'}e^{f_\theta(s)_{a'}}}, \tag{6}$$

where $f_\theta(s)$ is the logit vector produced by the network.

The clipped surrogate objective remains unchanged:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{7}$$

but the probability ratio $r_t(\theta)$ simplifies to:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} = \frac{e^{f_\theta(s_t)_a - f_{\theta_{\text{old}}}(s_t)_a}}{\sum_{a'} e^{f_{\theta_{\text{old}}}(s_t)_{a'}}} \cdot \frac{\sum_{a'} e^{f_{\theta_{\text{old}}}(s_t)_{a'}}}{\sum_{a'} e^{f_\theta(s_t)_{a'}}}. \tag{8}$$

In practice, this is computed efficiently using log-probabilities to avoid numerical instability.

To encourage exploration in discrete spaces, entropy regularization is also used:

$$\mathcal{L}^{\text{ENT}}(\theta) = \mathbb{E}_t \left[\sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \right]. \tag{9}$$

This prevents premature convergence to sub-optimal deterministic policies.

We also use the vectorized environment implementation from StableBaselines to improve the sample efficiency and training speed of PPO. Vectorized environments enable parallel data collection by running multiple instances of the environment simultaneously.

Let N be the number of environment instances running in parallel. At each timestep, the agent receives a batch of states $\mathbf{s}_t = (s_t^1, \ldots, s_t^N)$ and takes actions $\mathbf{a}_t = (a_t^1, \ldots, a_t^N)$ where:

$$a_t^i \sim \pi_\theta(\cdot|s_t^i) \quad \forall i \in \{1, \ldots, N\} \tag{10}$$

The rest of the algorithm remained unchanged as computed actions and rewards are flattened across all environments and batches.