Multi-Agent Reinforcement Learning Benchmark on Highway Environment for Autonomous Driving

Charbel Abi Hana
M1 International Track in Electrical Engineering
Université Paris-Saclay
Paris, France
charbel-a-h@outlook.com

***Abstract*—The rise of deep learning paved the way for the development of deep Reinforcement Learning methods in control, navigation and autonomous driving. The domain of Reinforcement Learning (RL) has become a powerful learning framwork capable of learning complex policies in high-dimensional environments. In this research project we aim to benchmark some of the widely used deep reinforcement learning methods on a simulated multi-agent highway environment. We will benchmark two decentralized algorithms; IDQN and IPPO which will control autonmously driven vehicles in a high density and high speed highway full of road (human driven) vehicles.**

***Index Terms*—Reinforcement Learning, Autonomous Driving, IDQN, IPPO**

I. Introduction

Autonomous driving systems constitute of multiple perception level tasks that have now achieved high precision on account of deep learning architectures. Besides the perception, autonomous driving systems constitute of multiple tasks where classical supervised learning methods are no more applicable. First, when the prediction of the agent's action changes future sensor observations received from the environment under which the autonomous driving agent operates, for example the task of optimal driving speed in an urban area. Second, supervisory signals such as time to collision, lateral error with respect to optimal trajectory of the agent, represent the dynamics of the agent, as well uncertainty in the environment. Such problems would require defining the stochastic cost function to be maximized. Third, the agent is required to learn new configurations of the environment, as well as to predict an optimal decision at each instant while driving in its environment. This represents a high dimensional space given the number of unique configurations under which the agent and environment are observed, this is combinatorially large. In such cases, we will solve the decision making process by formalizing it under the classical settings of Reinforcement Learning where an agent is required to optimally act in a given environment given a representation of that environment at a certain time step. The optimal set of actions taken by the agent is called the policy. In this research project, we will define the basic building blocks of a Reinforcement Learning algorithm, provide two implementations of those algorithms which leverage deep learning through multi-layered perceptron or convolution neural networks like the **IDQN** and **IPPO** which are direct extensions of the well-known Deep-Q-Learning (DQN [1]) and Proximal Policy Optimization (PPO [2]) respectively on a multi-agent level. The performance of each algorithm will be tested on an OpenAI Gym environment called *Highway-env* which simulates real world driving on an n-lane highway.

II. Background

A. Reinforcement Learning

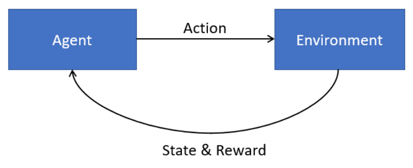
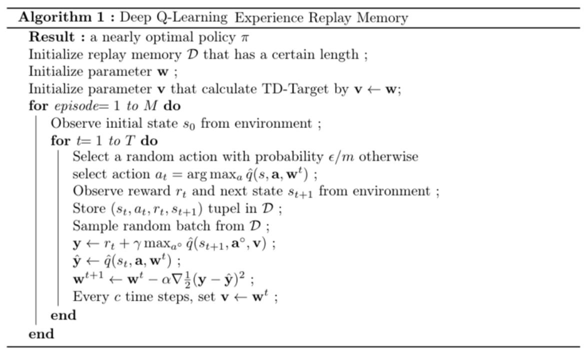


Fig. 1. Generated Output after 50 Epochs

B. Multi-Agent Reinforcement Learning

C. Deep Q-Learning



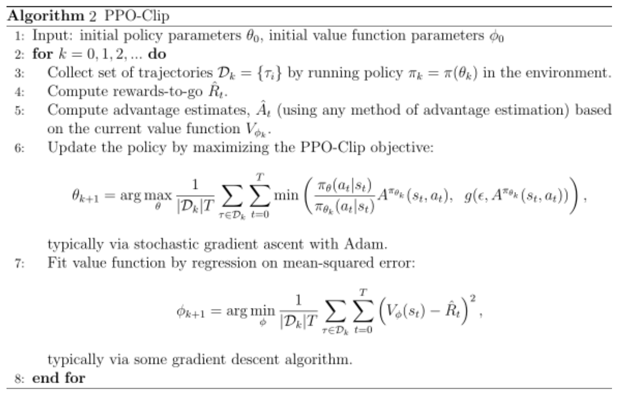
```
Algorithm 1 : Deep Q-Learning Experience Replay Memory
Result : a nearly optimal policy $\pi$
Initialize replay memory $\mathcal{D}$ that has a certain length ;
Initialize parameter $\mathbf{w}$ ;
Initialize parameter $\mathbf{v}$ that calculate TD-Target by $\mathbf{v} \leftarrow \mathbf{w}$;
for $episode$= 1 to $M$ do
    Observe initial state $s_0$ from environment ;
    for $t$= 1 to $T$ do
        Select a random action with probability $\epsilon/m$ otherwise
        select action $a_t = \arg\max_a \hat{q}(s, \mathbf{a}, \mathbf{w}^t)$ ;
        Observe reward $r_t$ and next state $s_{t+1}$ from environment ;
        Store $(s_t, a_t, r_t, s_{t+1})$ tupel in $\mathcal{D}$ ;
        Sample random batch from $\mathcal{D}$ ;
        $\mathbf{y} \leftarrow r_t + \gamma \max_{a^\circ} \hat{q}(s_{t+1}, \mathbf{a}^\circ, \mathbf{v})$ ;
        $\hat{\mathbf{y}} \leftarrow \hat{q}(s_t, \mathbf{a}, \mathbf{w}^t)$ ;
        $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \alpha \nabla \frac{1}{2}(\mathbf{y} - \hat{\mathbf{y}})^2$ ;
        Every $c$ time steps, set $\mathbf{v} \leftarrow \mathbf{w}^t$ ;
    end
end
```

Fig. 2. Deep Q-Learning Algorithm

D. Proximal Policy Optimization

$$\min_G \max_D \mathcal{V}_{\text{GAN}}(D, G) = \tag{1}$$

$$\mathbb{E}_{x \sim p_{\text{data}(x)}} [\log \{D(x)\}] + \mathbb{E}_{z \sim p_z(z)} [\log \{1 - D(G(z))\}] .$$



```
Algorithm 2 PPO-Clip
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, \ldots$ do
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
       on the current value function $V_{\phi_k}$.
6:     Update the policy by maximizing the PPO-Clip objective:
           $\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t))\right),$
       typically via stochastic gradient ascent with Adam.
7:     Fit value function by regression on mean-squared error:
           $\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2,$
       typically via some gradient descent algorithm.
8: end for
```

Fig. 3. Proximal Policy Optimization Algorithm

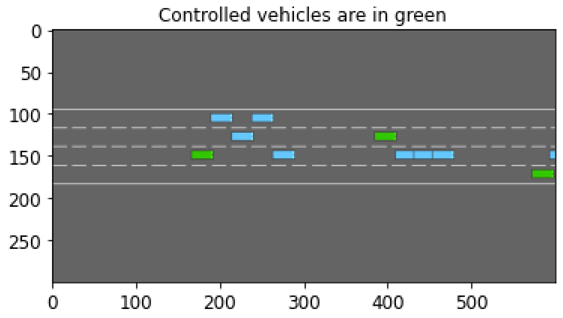


Fig. 4. Random Cows and Horses Images from Original Dataset

where $G : R^{100} \longrightarrow R^{16,384}$

$$L_D = -\sum_{x \in \chi, z \in \zeta} \log(D(x)) + \log(1 - D(G(z))) \tag{6}$$

$$L_G = -\sum_{z \in \zeta} \log(D(G(z)) \tag{7}$$

$$h^{[i]} = LeakyRELU(W^{[i-1]} h^{[i-1]} + b[i-1]) \tag{2}$$

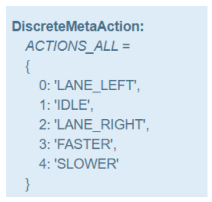
α, with h[i] $\epsilon R^{16\alpha 2^i}$ and we output the vector o $\epsilon R^{16,384}$ via

$$o = \tanh(W^{[L]} h^{[L]} + b[L]) \tag{3}$$

Where L is the final layer.

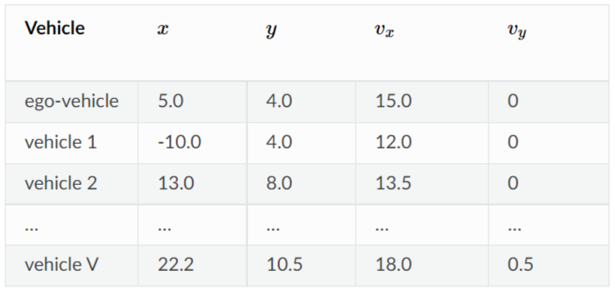
$h[0]$ denote the input image, $W[j]$ and $b[j]$ denoting the weight matrix and the bias vector in the L output layer, we have:

$$o = sigmoid(W^{[L]} h^{[L]} + b[L]) \tag{4}$$



```
DiscreteMetaAction:
  ACTIONS_ALL =
  {
    0: 'LANE_LEFT',
    1: 'IDLE',
    2: 'LANE_RIGHT',
    3: 'FASTER',
    4: 'SLOWER'
  }
```

Fig. 5. Generated Output after 100 Steps



Vehicle	x	y	v_x	v_y
ego-vehicle	5.0	4.0	15.0	0
vehicle 1	-10.0	4.0	12.0	0
vehicle 2	13.0	8.0	13.5	0
...	...	...	...	...
vehicle V	22.2	10.5	18.0	0.5

Fig. 6. Generated Output after 35000 Steps

Metric	Orig		Orig-250-GAN		Orig-500-GAN		Orig-1000-GAN	
-	**train**	**val**	**train**	**val**	**train**	**val**	**train**	**val**
BCE-Loss	0.01	5	0.01	0.7	0.01	0.7	8	0.7
Accuracy	1.0	0.5	1.0	0.6	1.0	0.5	0.5	0.5
Precision	1.0	0.1	1.0	0.5	1.0	0.5	0.5	0.5
Recall	1.0	0.5	1.0	0.4	1.0	1.0	1.0	1.0

TABLE I
COMPILED TABLE OF CLASSIFICATION METRICS

III. ENVIRONMENT

A. *Action Space*

B. *Observation Space*

C. *Reward*

IV. EXPERIMENTATION AND RESULTS

A. *Evaluation Metrics*

B. *Results*

V. CONCLUSION

REFERENCES

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.