

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 framework 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 autonomously 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 [2]) and Proximal Policy Optimization (PPO [4]) 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

Reinforcement Learning is a Machine Learning discipline where it uses many well-established methods of supervised learning such as deep neural networks for function approximation, stochastic gradient descent and backpropagation to learn data representations.

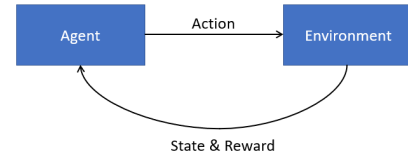


Fig. 1. Generic Reinforcement Learning algorithm diagram

Reinforcement learning models learn from an environment. The environment has a set of rules and is usually assumed to be deterministic. A reinforcement learning model interacts with the environment through an agent. The agent has a state in the environment, and the agent can perform actions that change the state of the agent in the environment. When the agent takes an action, the environment will receive this as an input and will output the resulting state and a reward (see diagram above). In other machine learning problems we usually start by defining a loss function, then we look to optimize. In reinforcement learning, we cannot immediately do that. To help us formulate a loss, we can start by looking at the rewards given back by the environment. An agent interacts with an environment through actions, these actions change the state of the environment. The goal of the model is to determine what actions will lead to the maximum reward.

To determine the best action, reinforcement learning works by estimating the value of actions. The value of an action indicates how good an action is. The value of an action

is defined as the sum of the immediate reward received by taking an action plus the expected value of the resulting state multiplied by a scaling term. In other words, the value of an action is how good the next state will be after taking that action, plus the expected future reward from that new state. Reinforcement learning models update their value function by interacting with the environment, choosing an action, looking at the new state, looking at the reward then updating.

Aside from the value function, the model needs to learn a policy. The policy of the algorithm is how it chooses what action to take based on the value of the current state. Reinforcement learning algorithms want to evaluate states as best as possible (value function) to help them make decisions (policy) that lead to the maximum reward.

B. Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning studies how multiple agents interact in a common environment. That is, when these agents interact with the environment and one another, can we observe them collaborate, coordinate, compete, or collectively learn to accomplish a particular task. It can be further broken down into three broad categories:

- Cooperative: All agents working towards a common goal
- Competitive: Agents competing with one another to accomplish a goal

Some mix of the two In addition, we can represent multi-agent problems in a centralized algorithm that communicates with the agents in the environment making all of their respective observations constantly available to the central algorithm. Another approach, which has shown more promise, is to consider each agent as an independent agent and each agent learns based on the reward function which deviates the training to a shared objective.

C. Q-Learning/Deep Q-Learning

Reinforcement learning is all about the value function (how good our state/actions are). The models learn sequentially by integrating with the environment for a number of episodes. Episodes can be thought of as epochs in reinforcement learning, and in the chess example they would indicate the number of complete games the agent trains for.

1) *Tabular Q-Learning*: Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

The q in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

When q-learning is performed we create what's called a q-table or matrix that follows the shape of [state, action] and we initialize our values to zero. We then update and store our q-values after an episode. This q-table becomes a reference table for our agent to select the best action based on the q-value.

An agent interacts with the environment in 1 of 2 ways. The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as exploiting since we use the information we have available to us to make a decision.

The second way to take action is to act randomly. This is called exploring. Instead of selecting actions based on the max future reward we select an action at random. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process. We can balance the exploration/exploitation by adjusting the discount factor shown in the formula below:

$$Q^{new}(s_t, a_t) \leftarrow Q^{old}(s_t, a_t) + \alpha * (r_t + \lambda * \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (1)$$

Where,

- $Q^{old}(s_t, a_t)$ is the old value
- α is the learning rate
- r_t is the current reward
- λ is the discount factor
- $\max_a Q(s_{t+1}, a)$ is the estimate of optimal future value

2) *Deep Q-Learning*: The Q-learning method that we've reviewed above solves the issue with iteration over the full set of states in the environment, but still can struggle with situations when the count of the observable set of states is very large. As a solution to this problem, we can use a nonlinear representation that maps both state and action onto a value. In machine learning this is called a **regression problem**. The concrete way to represent and train such a representation can vary, but, we will be using a deep neural network with Convolutional layers and Dense layers. We will be building on top of a DQN implementation developed by PFRL [1]. The algorithm for the Deep Q-learning method is shown below.

Algorithm 1 : Deep Q-Learning Experience Replay Memory

Result : a nearly optimal policy π
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 ϵ/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}) tuple in \mathcal{D} ;
 Sample random batch from \mathcal{D} ;
 $\mathbf{y} \leftarrow r_t + \gamma \max_a \hat{q}(s_{t+1}, \mathbf{a}, \mathbf{v})$;
 $\hat{\mathbf{y}} \leftarrow \hat{q}(s_t, \mathbf{a}, \mathbf{w}^t)$;
 $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \alpha \nabla_{\frac{1}{2}} (\mathbf{y} - \hat{\mathbf{y}})^2$;
 Every c time steps, set $\mathbf{v} \leftarrow \mathbf{w}^t$;
 end
end

Fig. 2. Deep Q-Learning Algorithm

D. Proximal Policy Optimization

Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm. The optimization objective of Policy Gradient methods defined as following:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \quad (2)$$

Where,

- $L^{PG}(\theta)$ is the policy loss
- $\log \pi_\theta(a_t|s_t)$ is the log of the probabilities from the output of the policy network
- \hat{A}_t is the estimate of the relative value of selected action.

The policy π_θ is the neural network that takes the state observation from an environment as input and suggests actions to take as an output. The advantage is an estimation, hence the hat over A, of the relative value for selected action in the current state. It is computed as a discounted reward (Q) — value function, where the value function basically gives an estimate of discounted sum of reward. When training, this neural net representing the value function will frequently be updated using the experience our agent collects in an environment. As much appealing it is to constantly perform gradient descent steps in one batch of collected experience, it will often update the parameters so far outside of the range that leads to **destructively large policy updates**.

1) *Trust Region Policy Optimization*: One of the approaches to prevent such destructive policy updates was Trust Region Policy Optimization [3]. In this paper, the authors implemented an algorithm to limit the policy gradient step so it does not move too much away from the original policy, causing overly large updates that often ruin the policy altogether.

2) *Clipped Surrogate Objective*: Proximal Policy Optimization attempts to simplify the optimization process while retaining the advantages of TRPO. One of this paper's main contribution is the clipped surrogate objective:

$$L^{PG}(CLIP) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad (3)$$

Here, the expectation over a minimum of two terms is computed: normal PG objective and clipped PG objective. The key component comes from the second term where a normal PG objective is truncated with a clipping operation between 1-epsilon and 1+epsilon, epsilon being the hyperparameter. The final loss function by summing this clipped PPO objective and two additional terms:

$$L^{PG}(CLIP+VF+S) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \quad (4)$$

The c_1 and c_2 are hyperparameters. The first term is a mean square error of the value function in charge of updating the baseline network. The second term, which may look unfamiliar, is an entropy term used to ensure enough exploration for our agents. This term will push the policy to behave more spontaneously until the other part of the objective starts dominating. The algorithm altogether for multiple epoch updating is shown below.

III. ENVIRONMENT

The environment used in this research project is *highway-env*

Algorithm 2 PPO, Actor-Critic Style

```

for iteration=1,2,... do
  for actor=1,2,...,N do
    Run policy  $\pi_{\theta_{old}}$  in environment for  $T$  timesteps
    Compute advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
  end for
  Optimize surrogate  $L$  wrt  $\theta$ , with  $K$  epochs and minibatch size  $M \leq NT$ 
   $\theta_{old} \leftarrow \theta$ 
end for

```

Fig. 3. Proximal Policy Optimization Algorithm

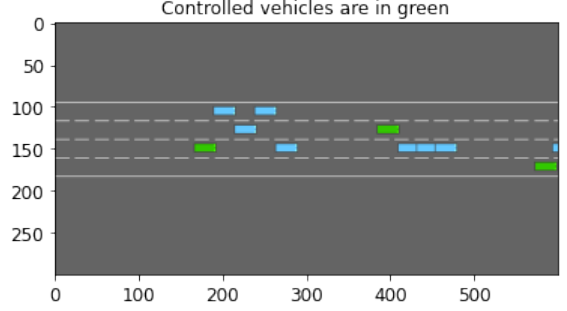


Fig. 4. Random Cows and Horses Images from Original Dataset

A. Action Space

B. Observation Space

C. Reward

IV. EXPERIMENTATION AND RESULTS

A. Evaluation Metrics

B. Results

V. CONCLUSION

REFERENCES

- [1] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. Chainerrl: A deep reinforcement learning library. *Journal of Machine Learning Research*, 22(77):1–14, 2021.

```

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

- [2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
- [3] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015.
- [4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.