A Comparison Between Deep Q-Learning and Actor-Critic in the CartPole Environment

Giulio Vaccari

Autonomous and Adaptive Systems 2020/2021

06/2021

Abstract

This report will analyze the behaviour of two different reinforcement learning algorithms tested in the CartPole environment provided by Open AI Gym. The first approach uses a Deep Q-Network to model a DQN agent, which follows an ϵ-greedy policy to choose each turn its next move. The second approach uses a more complex Actor-Critic architecture, considered in a one-step variant.

1 Introduction

Action-Value methods like DQN (Deep Q-Network) are based on the estimation of a value function used to infer the goodness of a possible move taken from a specific environment state. In this family of reinforcement learning algorithms, the value function is directly used by the agent's policy to determine which move to take. This approach is conceptually easy, and produced good results in different fields. However, action-value methods present some limitations, which are principally due to the policy used by the agent which exploits the learned value function. A different family of reinforcement learning algorithms is represented by the Policy Gradient Methods, which are based on the estimation of a parametrised policy without using an explicit value function. These algorithms present different advantages over the action-value ones, which include the possibility for the agent to learn a stochastic policy, and the continuous nature of the learned policy which changes smoothly as a function of the learned parameters. The combination of action-value and policy gradient techniques gave birth to a new family of algorithms called Actor-Critic, which are based on learning at the same time a value function and a parametrised policy. In the Actor-Critic architecture the final goal is to learn a parametrised policy like in the Policy Gradient methods, but the training process involves the learning of

a state-value function which is used in the update process of the policy at each training step.

2 Chosen Environment

For this comparison we chose the CartPole-v1 environment provided by Open AI Gym. In this classic 2D control problem we have a pole connected to a mobile cart which is controlled by the agent. The goal for the agent is to learn how to move the cart in order to balance the pole, preventing it from falling over. An environment state is represented by four components which are: cart position, cart velocity, pole angle, and the pole velocity at tip. In each time step the agent has two actions available which consist in the possibility to push the cart either to the right or to the left. Finally, the reward mechanism is designed to reward the ability of the agent to maintain the equilibrium for as much time as possible: reward is +1 for each time step played until the pole falls or until the time limit for an episode, which is equal to 500 time steps.

3 DQN Agent

The goal of the DQN agent is to learn a value function to estimate which are the most promising moves to take from each possible game state. To represent the value function, it has been used a deep neural network, while the policy used by the agent was an ϵ-greedy.

3.1 Value Network

The neural network used to represent the value function (Value Network Q) consists in 3 dense layers which map an environment observation to two Q-values, one for each possible action. The first dense layer maps a 4-dimensional observation to a hidden representation by using 128 hidden units and a ReLu activation function. The second dense layer consists of 64 hidden units with ReLu activation function, while the last layer has 2 units to represent the two actions, using a linear activation function.

3.2 Loss Function

The loss function for the Value Network was designed to improve its ability to estimate the quality of a possible move. The updates of the network follow a time-difference methodology, where at each time step t we adjust the prediction $Q(S_t, A_t)$ of the value function by looking only at the next observed reward R_{t+1} and at the values $Q(S_{t+1}, a)$ for each possible move a. Basically, at each training step we minimize the Mean Squared Error

(MSE) between the predicted Q-values for the actions at time t and the target Q-values computed by using information obtained at time step $t+1$:

$$Predicted\ Values = Q(S_t, A_t)$$

$$Target\ Values = \begin{cases} R_{t+1} + \gamma \max_a Q(S_{t+1}, a) & \text{if } S_t \text{ is not a termination state} \\ R_{t+1} & \text{otherwise} \end{cases}$$

$$DQN\ Loss = MSE(Target\ Values, Predicted\ Values)$$

Where γ is a discount factor used to weight the predicted future reward.

4 Actor-Critic Agent

The Actor-Critic agent uses two different neural networks during its training: a value network used to estimate how good a state is (Critic Network), and a policy network that will define the behaviour of the agent (Actor Network). The Critic network is used only in the training process, and it's designed to guide the updates of the policy network. The Actor network will be used also in the agent evaluation. At each time step of execution, the action performed by the agent is sampled according to the probability distribution $\pi(A|S)$ produced by the Actor network, conditioned by the current environment state S.

4.1 Critic Network

The Critic Network is implemented with a deep neural network which has the same 3-dense-layers architecture of the DQN agent. The only difference is in the final layer: while the DQN's network outputs a value for each possible agent action, the Critic network outputs only one value instead, which represents the expected future reward starting from the state given as input to the network.

4.2 Actor Network

The Actor network is a deep neural network which maps an environment state to a probability distribution over the actions available to the agent. The network consists of 3 dense layers: The first two respectively with 128 and 64 units and ReLu activation functions, and the last one with two units (one for each possible action) and a softmax activation function.

4.3 Loss Functions

As for the updates of the DQN agent, also in this case we will follow a time-difference methodology considering only two subsequent time steps in each update. This time we have two loss functions instead of one, which reflects

the two-network architecture of the Actor-Critic agent. The loss function associated to the Critic network $V(S)$ is designed to update the predicted value for the current state $V(S_t)$ by considering the reward R_{t+1} plus the predicted $V(S_{t+1})$ as target:

$$
\begin{aligned}
Predicted\ Values &= V(S_t) \\
Target\ Values &= \begin{cases} R_{t+1} + \gamma V(S_{t+1}) & \text{if } S_t \text{ is not a termination state} \\ R_{t+1} & \text{otherwise} \end{cases} \\
Critic\ Loss &= MSE(Target\ Values, Predicted\ Values)
\end{aligned}
$$

The loss function associated with the Actor network $\pi(A|S)$ has to change the probability distribution of the actions according to how well the actions performed in the past. The Actor's loss function uses the values produced by the Critic network to decide the magnitude and the direction of the update vector during a training step:

$$
\begin{aligned}
Critic\ Component &= \begin{cases} R_{t+1} + \gamma V(S_{t+1}) - V(S_t) & \text{if } S_t \text{ is not a termination state} \\ R_{t+1} - V(S_t) & \text{otherwise} \end{cases} \\
Actor\ Loss &= -\ln \pi(A_t|S_t) \cdot Critic\ Component
\end{aligned}
$$

5 Training

For both agents the training was performed using an experience replay buffer which stored information regarding single transitions between two states at subsequent time steps. The DQN agent used a buffer of size 2000, while the Actor-Critic agent employed a smaller buffer of size 500. While it's common to have a large replay buffer for a Q-Learning algorithm, for Actor-Critic the choice of a small buffer allowed to train the agent only on recent experiences, which helped to make the training process converge according to the Policy Gradient Theorem. The agents were trained for 400 episodes, after some initial warm-up episodes to partially fill the replay buffers. Each training step used a batch of 32 experiences sampled from the buffers. The value network of the DQN agent and the two networks of the Actor-Critic Agent used Adam as optimizer, but they differed in the choice of the learning rates. The DQN network used a learning rate of $1e-3$, while for the Actor-Critic agent, it has been used a learning rate of $1e-3$ for the Critic and of $1e-5$ for the Actor. It has been observed that is appropriate to have a learning rate for the Actor network one or two orders of magnitude lower compared to the one of the Critic network.

The DQN agent required an additional parameter which was the ϵ to be used in its ϵ-greedy policy, that needed to be adjusted during training to obtain a good compromise between exploration and exploitation. Starting

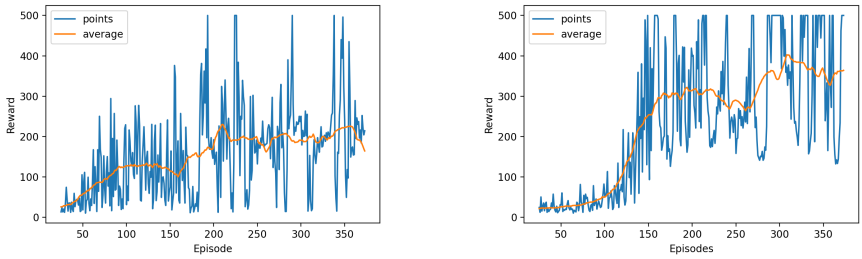
from an ϵ value of 1, its value decreases linearly until reaching the value of 0.05 at 6/10 of the episodes, and then remaining constant. Lastly, for both the agents was chosen a discount factor γ equals to 0.95.

6 Evaluation

For the evaluation of the two agents after the training, it has been averaged their score on 20 simulations for each one. During the evaluation, the DQN agent used an ϵ value of 0.05.

7 Results

From the results we can see that both agents performed well in the balancing task of CartPole, but the Actor-Critic agent performed considerably better. Looking at the average scores between 20 simulations, the DQN agent achieved a score of **181.75**, while the Actor-Critic agent a score of **488.1**. The comparison between the training progress of the agents also showed an higher stability over time in the results achieved by Actor-Critic.



(a) DQN (b) Actor-Critic

Figure 1: Total reward over training episodes for the two agents. The average was computed with a sliding window of 50 observations.

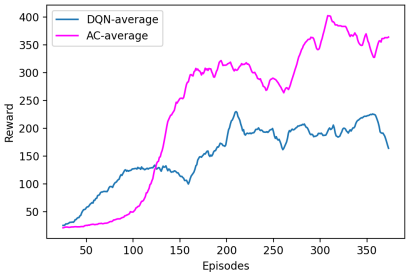
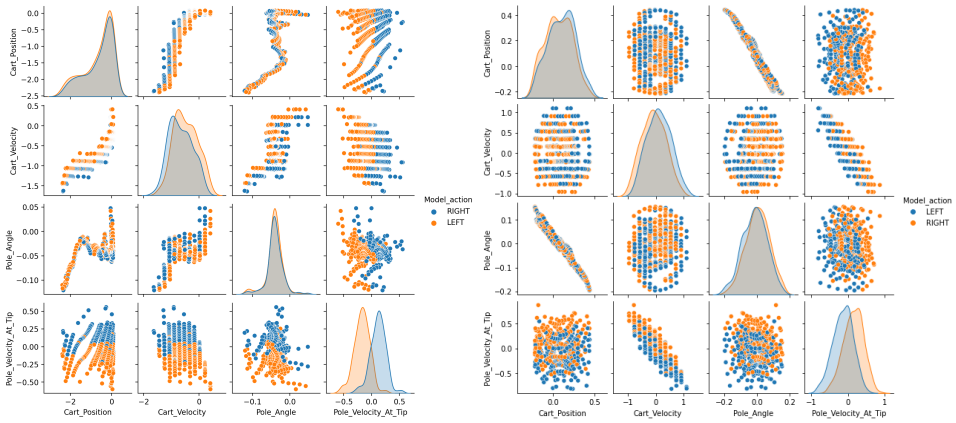


Figure 2: Comparison of the reward over training episodes for the two agents.

The pair plots can provide an insight at the reasoning process carried out by the agents when choosing a move given an input state. For example, we can see how for both agents the pole velocity at tip was an important feature to determine the action to perform.



(a) DQN (b) Actor-Critic

Figure 3: Pair plots for the two agents computed on 500 samples.

8 Conclusion

Both agents were capable of achieving good scores in the CartPole problem. Actor-Critic eventually obtained an higher score compared to DQN, but the fluctuations registered in the episode rewards during training showed some instabilities present in both algorithms, in particular for DQN. There were different advantages provided by the AC policy compared to the ϵ-greedy used by Q-Learning: the fact of not having to tune an ϵ parameter, together with the fast convergence to a good score, are great points in favor of this approach.