[proposition]Definition

DOC Q-LEARNING

Contents

1 Reinforcement Learning formalism

A reinforcement learning problem is formalized by an agent (our algorithm) which learns in an environment modeled by a set of states. At each iteration, it makes choices (actions) that will bring it to a state, and receives a reward. The goal is then to maximize the final expected reward. More concretely we suppose that the state process is controlled by a markovian $\mathbb{F}$-adapted process $(a_t)_{t\in\mathbb{T}} \in \mathcal{A}$ either discrete or continuous, namely for all $t \in \mathbb{T}$:

$$X_{t+1} = F(X_t, \epsilon_t, a_t) \in \mathcal{S} \subset \mathbb{R}^d \tag{1}$$

where $(\epsilon_t)_{t\in\mathbb{T}}$ are random noises.

(Markov Decision Process)
A Markov Decision Process (MDP) is quadruplet $(\mathcal{S}, \mathcal{A}, P, r)$ where :

- $\mathcal{S}$ and $\mathcal{A}$ are respectively the state space and action space (introduced above).
- P is the Markov kernel of $(X_t)_{t\in\mathbb{T}}$, namely the probability of transitioning from states given an action. We note $P(s'|(s,a)) := \mathbb{P}(X_{t+1} = s'|X_t = s, a_t = a)$ the probability that the next state is s' knowing that we are at step s and have taken action s.
- r is the reward function, how an agent get a feedback from its actions :

$$r : \begin{cases} \mathcal{S} \times \mathcal{A} \times \mathcal{S} & \rightarrow & R \\ (s, a, s') & \mapsto & r(s, a, s') \end{cases} \tag{2}$$

  (where R is the reward space either a discrete or continuous set). The quantity $r(s, a, s')$ is the reward obtained in state s', after taking action a in state s (as the state process is markovian so is the reward process).

Let $\mathcal{P}(\mathcal{A})$ be the set of all probability density functions if $\mathcal{A}$ if is continuous or mass functions if $\mathcal{A}$ is discrete. At each time $t_k \in \mathbb{T}$ we assume that the actions are chosen given a policy π:

(randomized policy)
π is called a randomized policy when it is a measurable transition kernel :

- $$\pi : (t, x) \in \mathbb{T} \times \mathcal{S} \rightarrow \pi(.|(t, x)) \in \mathcal{P}(\mathcal{A})$$
- $$(a, t, x) \in \mathcal{A} \times \mathbb{T} \times \mathcal{S} \rightarrow \pi(a|(t, x)) \text{ is measurable}$$

(deterministic policy)
π is called a deterministic policy when it is a measurable map :

$$\pi : (t,x) \in \mathbb{T} \times \mathcal{S} \to \pi(t,x) \in \mathcal{A}$$

In practise the source of randomness from a randomized policy is independent from the probability of the state function. So to encompass this we assume as in [?] that the probability space is large enough to support a uniform random variable Z indepent of $(\epsilon_t)_{t\in\mathbb{T}}$, let us note $\mathbb{F} = (\mathcal{F}_n \cup \sigma(Z))_{n\in\mathbb{N}}$ and $\mathbb{P}^*$ the new filtration and probability. We note $(X_t^\pi)_{t\in\mathbb{T}}$ the process generated by the policy π and $\mathcal{P}$ the set of all randomized policies.

The agent seeks to maximize the total expected reward more precisely :

(value function and action-state function)
Let r_k be short for $r(X_{k+1}^\pi, a_k, X_k^\pi)$. Let $G_\pi(k) = \sum_{i\geq k}^{t_n} \gamma^i r_{i+1}$ where $\gamma \in (0,1)$ is an actualization factor. The value function and action-state function are respectively:

$$v_\pi(k,s) = \mathbb{E}_{\mathbb{P}^*}[G_\pi(k)|X_{t_k} = s]$$
$$Q_\pi(k,s,a) = \mathbb{E}_{\mathbb{P}^*}[G_\pi(k)|X_{t_k} = s, a_{t_k} = a]$$

The goal of the agent is to find the policy that maximises thoses function :

$$v_*(k,s) = \sup_{\pi\in\mathcal{P}} \mathbb{E}_{\mathbb{P}^*}[G_\pi(k)|X_{t_k} = s]$$
$$Q_*(k,s,a) = \sup_{\pi\in\mathcal{P}} \mathbb{E}_{\mathbb{P}^*}[G_\pi(k)|X_{t_k} = s, a_{t_k} = a]$$

Both functions are easily interpretable, the value function estimates how good it is to be in a state averaging over all possible actions , while the action-state function estimates how good it is to take a given action in a state. Using Dynamic Programming, we can write the Bellmann equations for the action-state function :

Theorem 1.1. (Bellman equations)
We have :

$$v_\pi(s) = \mathbb{E}_{\mathbb{P}^*}\left[r(X_{t_k+1}^\pi, a_{t_k}, x) + \gamma v_\pi(X_{t_k+1}^\pi)|X_{t_k} = x\right]$$
$$Q_\pi(k,x,a) = \mathbb{E}_{\mathbb{P}^*}[r(X_{t_k+1}^\pi, a, x) + \gamma \mathbb{E}_{\mathbb{P}^*}[Q_\pi(k+1, X_{t_k+1}^\pi, a_{t_k+1})|X_{t_k} = x, a_{t_k} = a]]$$

and :

$$v_*(s) = \mathbb{E}_{\mathbb{P}^*}\left[r(X_{t_k+1}^\pi, a_{t_k}, x) + \gamma \sup_{\pi\in\mathcal{P}} v_*(X_{t_k+1}^\pi)|X_{t_k} = x\right]$$
$$Q_*(k,x,a) = \mathbb{E}_{\mathbb{P}^*}[r(X_{t_k+1}^\pi, a, x) + \gamma \sup_{\pi\in\mathcal{P}} \mathbb{E}_{\mathbb{P}^*}[Q_*(k+1, X_{t_k+1}^\pi, a_{t_k+1})|X_{t_k} = x, a_{t_k} = a]$$

Proof.

$$
\begin{aligned}
Q_\pi(k,s,a) &= \mathbb{E}_{\mathbb{P}^*}[G_\pi(k)|X_{t_k}=s, a_{t_k}=a] \\
&= \mathbb{E}_{\mathbb{P}^*}[r(X^\pi_{t_{k+1}},a,s)+\gamma G_\pi(k+1)|X_{t_k}=s,a_{t_k}=a] \\
&= \mathbb{E}_{\mathbb{P}^*}\left[r(X^\pi_{t_{k+1}},a,s)+\gamma \mathbb{E}_{\mathbb{P}^*}[G_\pi(k+1)|X_{t_{k+1}},a_{t_{k+1}}]|X_{t_k}=s,a_{t_k}=a\right] \\
&= \mathbb{E}_{\mathbb{P}^*}[r(X^\pi_{t_k+1},a,s)+\gamma \mathbb{E}_{\mathbb{P}^*}[Q_\pi(k+1,X^\pi_{t_k+1},a_{t_k+1})|X_{t_k}=s,a_{t_k}=a]]
\end{aligned}
$$

where we have used in the third line the tower property. The proof for Q_* is identical and the value function is identitical. □

Note : When policy is deterministic the Bellman equation becomes:

$$Q_*(k,x,a) = \mathbb{E}_{\mathbb{P}}[r(X^\pi_{t_k+1},a,x)+\gamma \sup_{\tilde{a}\in\mathcal{A}} Q_*(k+1,X^\pi_{t_k+1},\tilde{a})|X_{t_k}=x,a_{t_k}=a]$$

2 Q-Learning Algorithm

The idea behind the Q-learning method is to approximate Q_* with a parametric function $Q(.,.,.,\theta)$ (for instance with MLP as in the previous sections). Note that only the continuation value is unknown, so we just have to learn $Q(.,.,1,\theta)$. The (deterministic) policy induced by this algorithm is simply:

$$\pi_\theta(t,s) = \arg\max_{a'\in\mathcal{A}} Q(t,s,a',\theta) \tag{3}$$

More specifically, θ is learned by minimizing the following cost function, which is derived from the Bellman equations.

(Loss function)

Let $\mathcal{B}$ be a batch observation $b=(t,s,a,s')\in\mathbb{T}\times\mathcal{S}\times\mathcal{A}\times\mathcal{S}$ of size B.

Define also :

$$Y_\theta(t,s,a,s') = (r(s',a,s)+\gamma \max_{a'\in\mathcal{A}} Q(s',t,a',\theta))$$

The the cost function is:

$$\mathcal{L}(\theta) = \frac{1}{B}\sum_{b\in\mathcal{B}} (Q(t,s,a,\theta)-Y_{\theta_k}(t,s,a,s'))^2$$

By optimizing this cost, the model adjusts θ to ensure that the parametric Q-function estimates becomes more consistent with the Bellman equations, thereby improving the accuracy over time.

In practise the quadruplets are sampled uniformly from the buffer $\mathcal{B}$, and not from π_θ as there can be redundancies or correlations among observations. As we only learn $Q(.,.,1,\theta)$ we do not retrieve the final reward that stops the process. Moreover it has been shown that using two separate networks for learning Q and Y_θ improves the convergence and stability (see [**?**]). We call

- $Q(.,.,.,\theta)$ the online network whose parameters change at every iterations.
- $Q(.,.,\tilde{\theta})$ the target network utilized for calculating the target $Y_{\tilde{\theta}}$, and whose parameters are copied from the online network every C iterations, where C is an hyperparameter.

Another crucial point which is crucial in reinforcement learning is how the agent retrieve the quadruplets. In the initial iterations if the observations are sampled directly from π_θ, the strategy would often be to stop immediately due to the agent's lack of knowledge of the environment. Therefore the agent must in a first time explore the environment. Below we define two types of explorations, the first one classical and the other introduced in [**?**]

(ϵ-greedy exploration)
Let ϵ and $\epsilon' > 0$. The agent choose an action randomly with probability $1-\epsilon$, else the action is taken from the policy π_θ ie : $a = argmax_{a' \in \mathcal{A}} Q(t,s,a',\theta)$ if we are in state s at time t.
At first, we take *epsilon* small to explore the environment, and then increase its value by ϵ' at each iteration to begin to learn the Q-function.

(exploration with gaussian noise)
Let σ and $\sigma' > 0$ and $\epsilon \sim \mathcal{N}(0,\sigma^2)$. Then in state s, at time t the action is taken from:

$$a = 1_{\left\{Q(t,s,1)-Q(t,s,0)+\epsilon \geq 0\right\}} \tag{4}$$

This decision is intuitively clear, from corollary 3.21 $1_{Q(t,s,1)-Q(t,s,0)\geq 0}$ is the optimal policy and here the decision is perturbed by a gaussian noise. As learning progresses, this variance is gradually reduced by a decay factor σ' at each iteration, allowing the agent to transition smoothly from exploration to exploitation.

We can sum up the general idea of this method in the following pseudo-code :

Hyperparameter	**Observed Effects**
ϵ/σ	A high or low value of this parameter may result in early exercise due to insufficient knowledge of the environment.
Number of Target Network Updates	Increasing this parameter slows convergence and increases the variance of the loss. However, a low value might cause instability.
Learning Rate	Reducing this parameter also slows convergence but makes it smoother.
Batch Size	Increasing this parameter decreases the variance of the price.

Table 1: Effects of Various Hyperparameters

Algorithm 1 Deep Q-Learning

Require: d dimension of the state process, N number of episodes, ν learning rate, C number of updates for target network, B size of batches.

get initial state s_0.
Initialize $\theta_0 \in \Theta_p^L$ and buffer $\mathcal{D}$.
for i in N **do**
    while episode is not done **do**
        Explore / Exploit (ϵ-greedy or with a gaussian noise)
        if a = 0 **then**
            stop.
        else
            Stock (s, a, r, s') in $\mathcal{D}$
        end if
    end while
    Sample B from $\mathcal{D}$
    Update θ_i with ADAM algorithm.
    Update target network every C iterations.
end for

References