

Reinforcement Learning

Lecture 3 : Reinforcement Learning Algorithms

Emilie Kaufmann



Université
de Lille



Ecole Centrale de Lille, 2021/2022

Reminder : Dynamic Programming

If the parameters of a Markov Decision Process (MDP) are known

- ▶ mean reward $(r(s, a))_{(s,a) \in \mathcal{S} \times \mathcal{A}}$
- ▶ transition probabilities $(p(s'|s, a))_{(s,a,s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}}$

one can compute the **optimal value** V^* and **optimal policy** π^* using the fact that they satisfy the **Bellman equations**.

- Finite horizon H : V_h^* and π_h^* for $h \in \{1, \dots, H\}$ computed using backwards induction from

$$V_h^*(s) = \max_a \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s'|s, a) V_{h+1}^*(s') \right]$$

- **Infinite horizon with discount factor** γ (our focus today) :
 π^* is stationary and

$$V^*(s) = \max_a \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^*(s') \right]$$

Reminder : Dynamic Programming

If the parameters of a Markov Decision Process (MDP) are known

- ▶ mean reward $(r(s, a))_{(s,a) \in \mathcal{S} \times \mathcal{A}}$
- ▶ transition probabilities $(p(s'|s, a))_{(s,a,s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}}$

one can compute the **optimal value** V^* and **optimal policy** π^* using the fact that they satisfy the **Bellman equations**.

- Finite horizon H : V_h^* and π_h^* for $h \in \{1, \dots, H\}$ computed using backwards induction from

$$V_h^*(s) = \max_a \left[r(s, a) + \sum_{s' \in \mathcal{S}} p(s'|s, a) V_{h+1}^*(s') \right]$$

- **Infinite horizon with discount factor** γ (our focus today) :
 π^* is stationary and

$$\forall s \in \mathcal{S}, V^*(s) = T^*(V^*)(s)$$

One may use **Value Iteration** or **Policy Iteration**

Reinforcement Learning

- ▶ $r(s, a)$ and $p(s'|s, a)$ are unknown, we can only **interact with the environment and observe transitions**

The RL interaction protocol :

$$\mathcal{H}_t = \sigma(s_1, a_1, r_1, s_2, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t)$$

denotes the history of observations up to the beginning of round t .

At each time t , the agent

- ▶ selects an action $a_t \sim \pi_t(s_t)$ according to some **behavior policy**

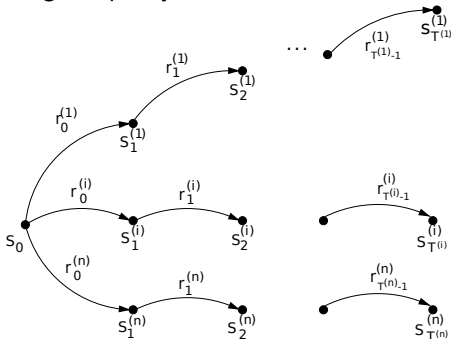
π_t may depend on \mathcal{H}_t

- ▶ observes the reward and next state

$$\begin{cases} r_t & \sim \nu_{(s_t, a_t)} \text{ such that } \mathbb{E}[r_t | s_t, a_t] = r(s_t, a_t) \\ s_{t+1} & \sim p(\cdot | s_t, a_t) \end{cases}$$

Reinforcement Learning

For example, starting from some state s_0 , one may observe several **trajectories** under a given policy.



One may also :

- ▶ restart in different states
- ▶ observe a single, very long, trajectory
- ▶ adaptively change the behavior policy

1 From Monte Carlo to Stochastic Approximation

2 Temporal Difference Learning for Policy Evaluation

3 Q-Learning for Finding the Optimal Policy

4 An Actor/Critic Variant

Monte Carlo estimation of a mean

A naive way to estimate a value is to use its definition as an expectation :

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_1 = s \right]$$

- Given n (long enough) trajectories under π starting from $s_1^{(i)} = s$,

$$t^{(i)} = (s_1^{(i)}, r_1^{(i)}, s_2^{(i)}, r_2^{(i)}, \dots, s_{T(i)}^{(i)}, r_{T(i)}^{(i)})$$

one can use the approximation

$$V^\pi(s) \simeq \frac{1}{n} \sum_{i=1}^n \underbrace{\left[\sum_{t=1}^{T(i)} \gamma^{t-1} r_t^{(i)} \right]}_{\text{i.i.d. with mean } \simeq V^\pi(s)}.$$

Properties

More generally, considering Z_i that are i.i.d. with mean μ , one can define the Monte-Carlo estimator

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n Z_i,$$

which has nice statistical properties, like $\hat{\mu}_n \xrightarrow{\text{a.s.}} \mu$.

Properties

More generally, considering Z_i that are i.i.d. with mean μ , one can define the Monte-Carlo estimator

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n Z_i,$$

which has nice statistical properties, like $\hat{\mu}_n \xrightarrow{\text{a.s.}} \mu$.

► Iterative rewriting

$$\hat{\mu}_n = \frac{n-1}{n} \hat{\mu}_{n-1} + \frac{1}{n} Z_n$$

Properties

More generally, considering Z_i that are i.i.d. with mean μ , one can define the Monte-Carlo estimator

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n Z_i,$$

which has nice statistical properties, like $\hat{\mu}_n \xrightarrow{\text{a.s.}} \mu$.

► Iterative rewriting

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \frac{1}{n} (Z_n - \hat{\mu}_{n-1})$$

Properties

More generally, considering Z_i that are i.i.d. with mean μ , one can define the Monte-Carlo estimator

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^n Z_i,$$

which has nice statistical properties, like $\hat{\mu}_n \xrightarrow{\text{a.s.}} \mu$.

► Iterative rewriting

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \alpha_n (Z_n - \hat{\mu}_{n-1})$$

for the **stepsize** $\alpha_n = \frac{1}{n}$.

→ Can we choose other stepsizes and still have $\hat{\mu}_n \xrightarrow{\text{a.s.}} \mu$?

Stochastic Approximation : Robbins-Monro

Goal : Find the solution to $\phi(x^*) = 0$ based on access to *noisy function evaluations*, i.e. for every x , one can observe a random value

$$Y = \phi(x) + \varepsilon,$$

where ε has zero mean (conditionally to previous queries).

Robbins-Monro algorithm (1951)

Given an initial x_0 , for all $n \geq 1$

- ▶ query a noisy evaluation $Y_n = \phi(x_{n-1}) + \varepsilon_n$
- ▶ update $x_n = x_{n-1} + \alpha_n Y_n$

Stochastic Approximation : Robbins-Monro

Goal : Find the solution to $\phi(x^*) = 0$ based on access to *noisy function evaluations*, i.e. for every x , one can observe a random value

$$Y = \phi(x) + \varepsilon,$$

where ε has zero mean (conditionally to previous queries).

Robbins-Monro algorithm (1951)

Given an initial x_0 , for all $n \geq 1$

- ▶ query a noisy evaluation $Y_n = \phi(x_{n-1}) + \varepsilon_n$
- ▶ update $x_n = x_{n-1} + \alpha_n Y_n$

Particular case : estimate a mean μ based on i.i.d. samples Z_i

$$\phi(x) = \mu - x \quad \text{and} \quad Y_n = Z_n - \hat{\mu}_{n-1}$$

Stochastic Approximation : Robbins-Monro

Goal : Find the solution to $\phi(x^*) = 0$ based on access to *noisy function evaluations*, i.e. for every x , one can observe a random value

$$Y = \phi(x) + \varepsilon,$$

where ε has zero mean (conditionally to previous queries).

Robbins-Monro algorithm (1951)

Given an initial x_0 , for all $n \geq 1$

- ▶ query a noisy evaluation $Y_n = \phi(x_{n-1}) + \varepsilon_n$
- ▶ update $x_n = x_{n-1} + \alpha_n Y_n$

Particular case : estimate a mean μ based on i.i.d. samples Z_i

$$\phi(x) = \mu - x \quad \text{and} \quad Y_n = Z_n - \hat{\mu}_{n-1}$$

Robbins-Monro update : $\hat{\mu}_n = \hat{\mu}_{n-1} + \alpha_n (Z_n - \hat{\mu}_{n-1})$.

Convergence of the Robbins-Monro algorithm

Theorem

Let $\phi : \mathcal{I} \subseteq \mathbb{R} \rightarrow \mathbb{R}$. Under the following assumptions

- ▶ ϕ is continuous and $\forall x \neq x^*, (x - x^*)\phi(x) < 0$
- ▶ there exists $C > 0$ such that $\mathbb{E}[Y_n^2 | x_{n-1}] \leq C(1 + x_{n-1}^2)$.
- ▶ the stepsizes satisfy

$$\sum_{n=1}^{\infty} \alpha_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty \quad (1)$$

under the Robbins-Monro algorithm, one has $x_n \xrightarrow{a.s.} x^*$.

Consequence : for the mean estimation problem, the sequence of iterates

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \alpha_n(Z_n - \hat{\mu}_{n-1})$$

converges almost surely to μ for **any stepsize α_n satisfying (1)** if $\mathbb{E}[Z_n^2 | X_{n-1}]$ is finite.

Robbins-Monro for fixed points

Goal : Find the solution to $x^* = T(x^*)$ based on access to **noisy** evaluations of $T(x)$.

Stochastic approximation for a fixed point

Given an initial x_0 , for all $n \geq 1$

- ▶ query a noisy evaluation $Z_n : \mathbb{E}[Z_n | x_{n-1}] = T(x_{n-1})$.
- ▶ update $x_n = x_{n-1} + \alpha_n (Z_n - x_{n-1})$

→ corresponds to the Robbins-Monro algorithm with

$$\phi(x) = T(x) - x \quad \text{and} \quad Y_n = Z_n - x_{n-1}.$$

1 From Monte Carlo to Stochastic Approximation

2 Temporal Difference Learning for Policy Evaluation

3 Q-Learning for Finding the Optimal Policy

4 An Actor/Critic Variant

Temporal Differences

Given a policy π , we want to compute V^π , which satisfies

$$V^\pi = T^\pi(V^\pi)$$

where $T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s')$.

- ▶ Given a current estimate \hat{V} , if we generate a trajectory under π

$$s_1, r_1, s_2, r_2, \dots, s_T, r_T,$$

one can produce **noisy evaluations** of $T^\pi(\hat{V})(s_k)$ for all $k \in \{1, \dots, T-1\}$ using

$$Z_k = r_k + \gamma \hat{V}(s_{k+1}).$$

$$\mathbb{E}[Z_k | \hat{V}, s_1, r_1, \dots, s_k] = r(s_k, \pi(s_k)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s_k, \pi(s_k)) \hat{V}(s')$$

Temporal Differences

Given a policy π , we want to compute V^π , which satisfies

$$V^\pi = T^\pi(V^\pi)$$

where $T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s')$.

- ▶ Given a current estimate \hat{V} , if we generate a trajectory under π

$$s_1, r_1, s_2, r_2, \dots, s_T, r_T,$$

one can produce **noisy evaluations** of $T^\pi(\hat{V})(s_k)$ for all $k \in \{1, \dots, T-1\}$ using

$$Z_k = r_k + \gamma \hat{V}(s_{k+1}).$$

$$\mathbb{E}[Z_k | \hat{V}, s_1, r_1, \dots, s_k] = T^\pi(\hat{V})(s_k)$$

Temporal Differences

Given a policy π , we want to compute V^π , which satisfies

$$V^\pi = T^\pi(V^\pi)$$

where $T^\pi(V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s')$.

- ▶ Given a current estimate \hat{V} , if we generate a trajectory under π

$$s_1, r_1, s_2, r_2, \dots, s_T, r_T,$$

one can produce noisy evaluations of $T^\pi(\hat{V})(s_k)$ for all $k \in \{1, \dots, T-1\}$ using

$$Z_k = r_k + \gamma \hat{V}(s_{k+1}).$$

- ▶ “Robbins-Monro” update : $\hat{V}(s_k) \leftarrow \hat{V}(s_k) + \alpha (Z_k - \hat{V}(s_k))$

Temporal Differences

Definition

The Robbins-Monro update rewrites

$$\hat{V}(s_k) \leftarrow \hat{V}(s_k) + \alpha \delta_k(\hat{V})$$

introducing the k -th **temporal difference** (or TD error) :

$$\delta_k(\hat{V}) := r_k + \gamma \hat{V}(s_{k+1}) - \hat{V}(s_k).$$

► Interpretation :

$$\delta_k(\hat{V}) := \underbrace{r_k + \gamma \hat{V}(s_{k+1})}_{\text{new estimate}} - \underbrace{\hat{V}(s_k)}_{\text{previous estimate}}$$

The value of the estimate is moved toward the value of the new estimate, which is itself built upon \hat{V} .

→ **Bootstrapping !**

Sutton, *Learning to Predict by the Method of Temporal Differences*, 1988

The TD(0) algorithm

Input : π : policy, T : number of iterations, $(\alpha_i(s))_{i \in \mathbb{N}}$: stepsizes,
 $V_0 \in \mathbb{R}^S$: initial values, $s_0 \in \mathcal{S}$: initial state (arbitrary)

```
1  $V \leftarrow V_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_S$ 
3 for  $t = 1, \dots, T$  do
4    $N(s) \leftarrow N(s) + 1$            \ \ update the number of visits of state  $s$ 
5    $(r, s') = \text{step}(s, \pi(s))$        \ \ perform a transition under  $\pi$ 
6    $V(s) \leftarrow V(s) + \alpha_{N(s)}(s) (r + \gamma V(s') - V(s))$ 
7    $s \leftarrow s'$ 
8 end
Return:  $V$ 
```

The TD(0) algorithm

Input : π : policy, T : number of iterations, $(\alpha_i(s))_{i \in \mathbb{N}}$: stepsizes,
 $V_0 \in \mathbb{R}^S$: initial values, $s_0 \in \mathcal{S}$: initial state (arbitrary)

```
1  $V \leftarrow V_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_S$ 
3 for  $t = 1, \dots, T$  do
4    $N(s) \leftarrow N(s) + 1$            \\ update the number of visits of state s
5    $(r, s') = \text{step}(s, \pi(s))$       \\ perform a transition under  $\pi$ 
6    $V(s) \leftarrow V(s) + \alpha_{N(s)}(s) (r + \gamma V(s') - V(s))$ 
7    $s \leftarrow s'$ 
8 end
Return:  $V$ 
```

$$(r, s') = \text{step}(s, \pi(s)) \Leftrightarrow \begin{cases} r & \sim \mathcal{V}_{(s, \pi(s))} \\ s' & \sim p(\cdot | s, \pi(s)) \end{cases}$$

The TD(0) algorithm

Input : π : policy, T : number of iterations, $(\alpha_i(s))_{i \in \mathbb{N}}$: stepsizes, $V_0 \in \mathbb{R}^S$: initial values, $s_0 \in \mathcal{S}$: initial state (arbitrary)

```
1  $V \leftarrow V_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_S$ 
3 for  $t = 1, \dots, T$  do
4    $N(s) \leftarrow N(s) + 1$            \\ update the number of visits of state s
5    $(r, s') = \text{step}(s, \pi(s))$        \\ perform a transition under  $\pi$ 
6    $V(s) \leftarrow V(s) + \alpha_{N(s)}(s) (r + \gamma V(s') - V(s))$ 
7    $s \leftarrow s'$ 
8 end
Return:  $V$ 
```

→ tuning the stepsizes?

The TD(0) algorithm

Theorem

It the step-size (also called *learning rate*) satisfy the Robbins-Monro conditions in all state s :

$$\sum_{i=1}^{\infty} \alpha_i(s) = +\infty \quad \text{and} \quad \sum_{i=1}^{\infty} (\alpha_i(s))^2 < +\infty$$

and *all states are visited infinitely often*, then

$$\lim_{T \rightarrow \infty} \hat{V}_T = V^\pi,$$

where \hat{V}_T denotes the output of TD(0) after T iterations.

► **Typical choice** : $\alpha_i(s) = \frac{1}{i^\beta}$ for $\beta \in (1/2, 1]$.

$$\hat{V}_t(s) = \hat{V}_{t-1}(s) + \frac{1}{N_t(s)^\beta} \left(r + \gamma \hat{V}_{t-1}(s') - \hat{V}_{t-1}(s) \right)$$

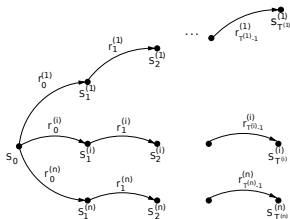
with $N_t(s)$ the number of visits of s up to the t -th iteration.

Monte-Carlo with Temporal Differences

Incremental Monte-Carlo for the estimation of

$$V^\pi(s_1) = \mathbb{E}^\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_1 \right]$$

based on n trajectories starting in s_1 :



Update after the i -th trajectory :

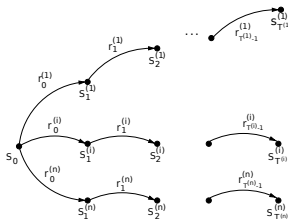
$$\hat{V}_i(s_1) = \hat{V}_{i-1}(s_1) + \alpha_i \left(\sum_{t=1}^{T^{(i)}} \gamma^{t-1} r_t^{(i)} - \hat{V}_{i-1}(s_0) \right)$$

Monte-Carlo with Temporal Differences

Incremental Monte-Carlo for the estimation of

$$V^\pi(s_1) = \mathbb{E}^\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_1 \right]$$

based on n trajectories starting in s_1 :



Update after the i -th trajectory : \rightarrow rewrites with the **temporal differences**

$$\hat{V}_i(s_1) = \hat{V}_{i-1}(s_1) + \alpha_i \left(\sum_{t=1}^{T^{(i)}-1} \gamma^{t-1} \delta_t^{(i)} (\hat{V}_{i-1}) + \gamma^{T^{(i)}-1} (r_{T^{(i)}}^{(i)} - \hat{V}_{i-1}(s_{T^{(i)}}^{(i)})) \right)$$

Monte-Carlo with Temporal Differences

$$\hat{V}_i(s_1) \simeq \hat{V}_{i-1}(s_1) + \alpha_i \left(\sum_{t=1}^{T^{(i)}-1} \gamma^t \delta_t^{(i)}(\hat{V}_{i-1}) \right)$$

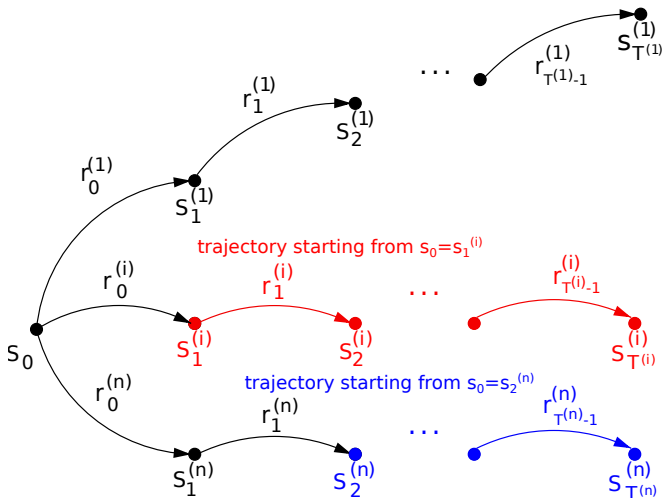
Limitation of naive Monte-Carlo :

- ▶ performing a full trajectory is needed before the update
- ▶ we only update the value of the initial state s_1

Extension :

- update the values of multiple states after each trajectory
- online updates, after each transition

Why update multiple states ?



Every visit Monte-Carlo

Every visits Monte-Carlo (a.k.a. **TD(1)**) : after the i -th trajectory, instead of updating only $\hat{V}(s_1)$, for all $k = T^{(i)} - 1$ down to 1,

$$\hat{V}(s_k^{(i)}) \leftarrow \hat{V}(s_k^{(i)}) + \alpha_i(s_k^{(i)}) \left(\sum_{t=k}^{T^{(i)}} \gamma^{t-k} r_t^{(i)} - \hat{V}(s_k^{(i)}) \right)$$

Remarks :

- ▶ multiple updates of states visited more than once in the trajectory
- ▶ **first visit** variant : update $s_k^{(i)}$ only is $s_k^{(i)} \notin \{s_1^{(i)}, \dots, s_{k-1}^{(i)}\}$

Every visit Monte-Carlo

Every visits Monte-Carlo (a.k.a. **TD(1)**) : after the i -th trajectory, instead of updating only $\hat{V}(s_1)$, for all $k = T^{(i)} - 1$ down to 1,

$$\hat{V}(s_k^{(i)}) \leftarrow \hat{V}(s_k^{(i)}) + \alpha_i(s_k^{(i)}) \left(\sum_{t=k}^{T^{(i)}} \gamma^{t-k} \delta_t^{(i)}(\hat{V}) \right)$$

Remarks :

- ▶ multiple updates of states visited more than once in the trajectory
- ▶ **first visit** variant : update $s_k^{(i)}$ only is $s_k^{(i)} \notin \{s_1^{(i)}, \dots, s_{k-1}^{(i)}\}$

TD methods for learning the optimal policy ?

TD methods permit to approximately compute V^π for a given policy π

→ can we use them to get to π^* ?

Hope : policy evaluation is a central ingredient in **Policy Iteration**

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 = \text{greedy}(V^{\pi_0}) \rightarrow V^{\pi_1} \rightarrow \pi_2 = \text{greedy}(V^{\pi_1}) \rightarrow V^{\pi_2} \rightarrow \dots \rightarrow \pi^*$$

TD methods for learning the optimal policy ?

TD methods permit to approximately compute V^π for a given policy π

→ can we use them to get to π^* ?

Hope : policy evaluation is a central ingredient in **Policy Iteration**

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 = \text{greedy}(V^{\pi_0}) \rightarrow V^{\pi_1} \rightarrow \pi_2 = \text{greedy}(V^{\pi_1}) \rightarrow V^{\pi_2} \rightarrow \dots \rightarrow \pi^*$$

Limitation : the policy improvement step cannot be performed without the knowledge of the MDP parameters

$$\begin{aligned}\pi_{k+1} &= \text{greedy}(V^{\pi_k}) \\ \Leftrightarrow \pi_{k+1}(s) &= \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^{\pi_k}(s') \right]\end{aligned}$$

TD methods for learning the optimal policy ?

TD methods permit to approximately compute V^π for a given policy π

→ can we use them to get to π^* ?

Hope : policy evaluation is a central ingredient in **Policy Iteration**

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 = \text{greedy}(V^{\pi_0}) \rightarrow V^{\pi_1} \rightarrow \pi_2 = \text{greedy}(V^{\pi_1}) \rightarrow V^{\pi_2} \rightarrow \dots \rightarrow \pi^*$$

Limitation : the policy improvement step cannot be performed without the knowledge of the MDP parameters

$$\begin{aligned} \pi_{k+1} &= \text{greedy}(V^{\pi_k}) \\ \Leftrightarrow \pi_{k+1}(s) &= \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^{\pi_k}(s') \right] \end{aligned}$$

Other possibility : work directly with Q-values !

- 1 From Monte Carlo to Stochastic Approximation
- 2 Temporal Difference Learning for Policy Evaluation
- 3 Q-Learning for Finding the Optimal Policy**
- 4 An Actor/Critic Variant

Reminder : Q-values

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V^\pi(s')$$

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$$

Properties

- ❶ Q^* satisfies the Bellman equations

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \max_{a' \in \mathcal{A}} Q^*(s', a')$$

- ❷ $V^*(s) = Q^*(s, \pi^*(s))$
- ❸ $\pi^* = \text{greedy}(Q^*)$, i.e. $\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$

→ New goal : Learning Q^*

A stochastic approximation scheme for Q^*

- ▶ Q^* also satisfies a fixed point equation : $Q^* = T^*(Q^*)$ where

$$T^*(Q)(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \max_{a' \in \mathcal{A}} Q(s', a').$$

- ▶ Noisy evaluations of $T^*(Q)(s_k, a_k)$ along a trajectory :

$$Z_k = r_k + \gamma \max_{a' \in \mathcal{A}} Q(s_{k+1}, a')$$

satisfies $\mathbb{E}[Z_k | \mathcal{H}_k, a_k] = T^*(Q)(s_k, a_k)$.

(for *any behavior policy*)

A stochastic approximation scheme for Q^*

- ▶ Q^* also satisfies a fixed point equation : $Q^* = T^*(Q^*)$ where

$$T^*(Q)(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) \max_{a' \in \mathcal{A}} Q(s', a').$$

- ▶ Noisy evaluations of $T^*(Q)(s_k, a_k)$ along a trajectory :

$$Z_k = r_k + \gamma \max_{a' \in \mathcal{A}} Q(s_{k+1}, a')$$

satisfies $\mathbb{E}[Z_k | \mathcal{H}_k, a_k] = T^*(Q)(s_k, a_k)$.

(for *any behavior policy*)

- **Robbins-Monro update :**

$$\hat{Q}(s_k, a_k) \leftarrow \hat{Q}(s_k, a_k) + \alpha \left(r_k + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s_{k+1}, a') - \hat{Q}(s_k, a_k) \right)$$

Q-Learning

Input : T : number of iterations, $(\alpha_i(s, a))_{i \in \mathbb{N}}$: step-sizes,
 $Q_0 \in \mathbb{R}^{S \times A}$: initial Q-values, $s_0 \in \mathcal{S}$: initial state (arbitrary)
 π_t : behavior policy

```
1  $Q \leftarrow Q_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_{S \times A}$ 
3 for  $t = 1, \dots, T$  do
4    $a \sim \pi_t(s)$  \ \ choose an action under the behavior policy
5    $N(s, a) \leftarrow N(s, a) + 1$  \ \ update the number of visits of (s, a)
6    $(r, s') = \text{step}(s, a)$  \ \ perform a transition
7    $Q(s, a) \leftarrow Q(s, a) + \alpha_{N(s, a)}(s, a) (r + \gamma \max_b Q(s', b) - Q(s, a))$ 
8    $s \leftarrow s'$ 
9 end
Return:  $Q, \pi = \text{greedy}(Q)$ 
```

[Watkins, 1989]

Q-Learning

Input : T : number of iterations, $(\alpha_i(s, a))_{i \in \mathbb{N}}$: step-sizes,
 $Q_0 \in \mathbb{R}^{S \times A}$: initial Q-values, $s_0 \in \mathcal{S}$: initial state (arbitrary)
 π_t : behavior policy

```
1  $Q \leftarrow Q_0, s \leftarrow s_0$ 
2  $N \leftarrow 0_{S \times A}$ 
3 for  $t = 1, \dots, T$  do
4    $a \sim \pi_t(s)$  \ \ choose an action under the behavior policy
5    $N(s, a) \leftarrow N(s, a) + 1$  \ \ update the number of visits of (s, a)
6    $(r, s') = \text{step}(s, a)$  \ \ perform a transition
7    $Q(s, a) \leftarrow Q(s, a) + \alpha_{N(s, a)}(s, a) (r + \gamma \max_b Q(s', b) - Q(s, a))$ 
8    $s \leftarrow s'$ 
9 end
Return:  $Q, \pi = \text{greedy}(Q)$ 
```

[Watkins, 1989]

Q-Learning

Theorem

It the step-size (also called *learning rate*) satisfy the Robbins-Monro conditions in all state action pair (s, a) :

$$\sum_{i=1}^{\infty} \alpha_i(s, a) = +\infty \quad \text{and} \quad \sum_{i=1}^{\infty} (\alpha_i(s, a))^2 < +\infty$$

and *all states-action pairs are visited infinitely often* , then

$$\lim_{T \rightarrow \infty} \hat{Q}_T = Q^*,$$

where \hat{Q}_T denotes the output of T iterations of Q-Learning.

Q-Learning

Theorem

It the step-size (also called *learning rate*) satisfy the Robbins-Monro conditions in all state action pair (s, a) :

$$\sum_{i=1}^{\infty} \alpha_i(s, a) = +\infty \quad \text{and} \quad \sum_{i=1}^{\infty} (\alpha_i(s, a))^2 < +\infty$$

and *all states-action pairs are visited infinitely often* , then

$$\lim_{T \rightarrow \infty} \hat{Q}_T = Q^*,$$

where \hat{Q}_T denotes the output of T iterations of Q-Learning.

→ typical step-sizes choice : $\alpha_i(s, a) = \frac{1}{i^\beta}$ with $\beta \in (1/2, 1]$.

Behavior Policy

- **Constraint** : all state-action pairs need to be visited infinitely often

$$\pi_t(s) = \mathcal{U}(\mathcal{S}) \rightarrow a_t \text{ chosen uniformly at random?}$$

- **Idea** : we care about π^* , we need to refine our estimate of Q^* in the pairs $(s, \pi^*(s))$ / we may want to maximize rewards while learning

$$\pi_t = \text{greedy} \left(\hat{Q}_{t-1} \right)?$$

ϵ -greedy exploration [Sutton and Barto, 1998]

The ϵ -greedy policy performs the following :

- with probability ϵ , select $a_t \sim \mathcal{U}(\mathcal{S})$
- with probability $1 - \epsilon$, select $a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \hat{Q}_t(s_t, a)$
- tends to the greedy policy when $\epsilon \rightarrow 0$

Behavior Policy

- **Constraint** : all state-action pairs need to be visited infinitely often

$$\pi_t(s) = \mathcal{U}(\mathcal{S}) \rightarrow a_t \text{ chosen uniformly at random?}$$

- **Idea** : we care about π^* , we need to refine our estimate of Q^* in the pairs $(s, \pi^*(s))$ / we may want to maximize rewards while learning

$$\pi_t = \text{greedy}(\hat{Q}_{t-1})?$$

Boltzmann (or softmax) exploration [Sutton and Barto, 1998]

The **softmax policy** with temperature τ is given by

$$(\pi_t(s))_a = \frac{\exp(\hat{Q}_t(s, a)/\tau)}{\sum_{a' \in \mathcal{A}} \exp(\hat{Q}_t(s, a')/\tau)}$$

and $a_t \sim \pi_t(s_t)$.

→ tends to the greedy policy when $\tau \rightarrow 0$

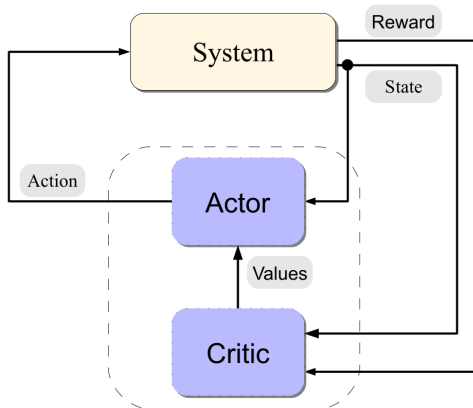
In practice

- ▶ Q-Learning (and more generally TD methods) can be very slow to converge...
- Let's try it on our Retail Store Management use case

- 1 From Monte Carlo to Stochastic Approximation
- 2 Temporal Difference Learning for Policy Evaluation
- 3 Q-Learning for Finding the Optimal Policy
- 4 An Actor/Critic Variant**

The Actor/Critic architecture

- ▶ **the actor** : update its policy to improve the value given by the critic
- ▶ **the critic** : evaluates the actor's policy



source : [Szepesvari, 2010]

Generalized Policy Iteration

Policy Iteration is an extreme example of an Actor/Critic architecture :

- ▶ **the actor** : “acts” with $\pi = \text{greedy}(V)$ where V is the value provided by the critic
- ▶ **the critic** : computes V^π where π is the current actor's policy

Generalized Policy Iteration

Policy Iteration is an extreme example of an Actor/Critic architecture :

- ▶ **the actor** : performs **policy improvement**
- ▶ **the critic** : performs **policy evaluation**
- Actor/Critic is also referred to as **Generalized Policy Iteration**

[Sutton and Barto, 1998]

There are many algorithms of this type !

An example : the SARSA algorithm

► The critic

After observing the actor's recent behavior ($s_t, a_t, r_t, s_{t+1}, a_{t+1}$), update

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left(r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t) \right)$$

State Action Reward State Action (SARSA) update

→ if the actor is following a fixed policy π ($a_t = \pi(s_t)$), SARSA=TD(0)

An example : the SARSA algorithm

► The critic

After observing the actor's recent behavior ($s_t, a_t, r_t, s_{t+1}, a_{t+1}$), update

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left(r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t) \right)$$

State Action Reward State Action (SARSA) update

→ if the actor is following a fixed policy π ($a_t = \pi(s_t)$), SARSA=TD(0)

► **The actor** : moves its behavior policy towards being greedy with respect to the Q -value provided by the critic, e.g.

→ ϵ -greedy policy

→ softmax policy with temperature τ

Q-Learning versus SARSA

The update rules of the two algorithms are close but not identical :

- ▶ **Q-Learning :**

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a') - \hat{Q}(s_t, a_t) \right)$$

- ▶ **SARSA :**

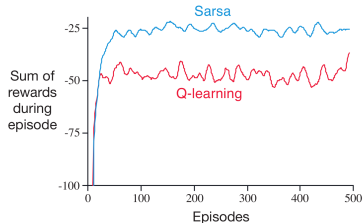
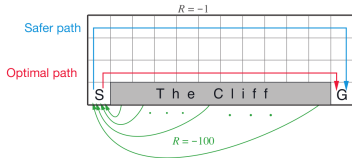
$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left(r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t) \right)$$

Both aim at learning the **target policy** $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$.

- ▶ Q-Learning converges for **any behavior policy** (exploring enough)
off-policy learning
- ▶ for SARSA the behavior policy is close to the estimated target policy
on-policy learning

Q-Learning versus SARSA

An example from [Sutton and Barto, 1998] : Q-Learning and SARSA used with ε -greedy exploration with $\varepsilon = 0.1$.



Observation : SARSA converges to a sub-optimal safer policy that yield more reward during learning, while Q-Learning converges to the optimal policy, while falling often from the cliff during learning

(if $\varepsilon \rightarrow 0$, SARSA would also converge to the optimal policy)



Sutton, R. and Barto, A. (1998).
Reinforcement Learning : an Introduction.
MIT press.



Sutton, R. S. (1988).
Learning to predict by the methods of temporal differences.
Machine Learning, 3 :9–44.



Szepesvari, C. (2010).
Algorithms for Reinforcement Learning.
Morgan & Claypool.



Watkins, C. (1989).
Learning from Delayed Rewards.
PhD thesis, University of Cambridge.