

Sequential Decision Making

Lecture 4 : Reinforcement Learning with Function Approximation

Rémy Degenne
(remy.degenne@inria.fr)



Université
de Lille



Centrale Lille, 2025/2026

Different Markov Decision Problems

Overall goal : learn **the optimal policy** π^* associated to some MDP parameterized by $r(s, a)$ and $p(\cdot|s, a)$ for $(s, a) \in \mathcal{S} \times \mathcal{A}$.

Different contexts :

- ① Small state space \mathcal{S} , known dynamics
- ② Small state space \mathcal{S} , unknown dynamics
- ③ Large state space \mathcal{S} , known dynamics
- ④ Large state space \mathcal{S} , unknown dynamics

Different Markov Decision Problems

Overall goal : learn the optimal policy π^* associated to some MDP parameterized by $r(s, a)$ and $p(\cdot|s, a)$ for $(s, a) \in \mathcal{S} \times \mathcal{A}$.

Different contexts :

- ① Small state space \mathcal{S} , known dynamics → Dynamic Programming
Value Iteration. Policy Iteration.
- ② Small state space \mathcal{S} , unknown dynamics
- ③ Large state space \mathcal{S} , known dynamics
- ④ Large state space \mathcal{S} , unknown dynamics

Different Markov Decision Problems

Overall goal : learn the optimal policy π^* associated to some MDP parameterized by $r(s, a)$ and $p(\cdot|s, a)$ for $(s, a) \in \mathcal{S} \times \mathcal{A}$.

Different contexts :

- ① Small state space \mathcal{S} , known dynamics → Dynamic Programming
Value Iteration. Policy Iteration.
- ② Small state space \mathcal{S} , unknown dynamics → Temporal Differences
Q-Learning. SARSA.
- ③ Large state space \mathcal{S} , known dynamics
- ④ Large state space \mathcal{S} , unknown dynamics

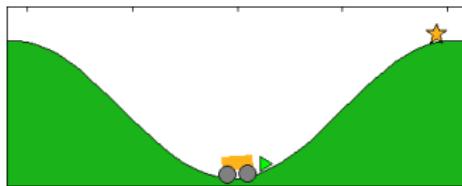
Different Markov Decision Problems

Overall goal : learn the optimal policy π^* associated to some MDP parameterized by $r(s, a)$ and $p(\cdot|s, a)$ for $(s, a) \in \mathcal{S} \times \mathcal{A}$.

Different contexts :

- ① Small state space \mathcal{S} , known dynamics → Dynamic Programming
Value Iteration. Policy Iteration.
- ② Small state space \mathcal{S} , unknown dynamics → Temporal Differences
Q-Learning. SARSA.
- ③ Large state space \mathcal{S} , known dynamics → ?
- ④ Large state space \mathcal{S} , unknown dynamics → ?

Example : Mountain Car



State : $(x, \dot{x}) \in [-1.2; 0.6] \times [-0.07; 0.07]$

Actions : $\mathcal{A} = \{-1, 0, 1\}$:

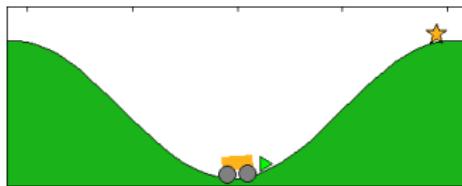
full speed backwards / do nothing / full speed forward

Reward : always -1 except in the **terminal (goal) state** $x_* = 0.6$

Dynamics : when doing action a_t in state $s_t = (x_t, v_t)$, the next state $s_{t+1} = (x_{t+1}, v_{t+1})$ is

$$\begin{cases} v_{t+1} &= \max\{\min\{v_t + \epsilon_t + 0.001a_t - 0.0025 \cos(3x_t), 0.07\}, -0.07\}, \\ x_{t+1} &= \max\{\min\{x_t + v_t, 0.6\}, -1.2\}. \end{cases}$$

Example : Mountain Car



State : $(x, \dot{x}) \in [-1.2; 0.6] \times [-0.07; 0.07]$

Actions : $\mathcal{A} = \{-1, 0, 1\}$:
full speed backwards / do nothing / full speed forward

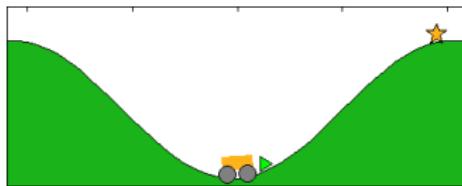
Reward : always -1 except in the terminal (goal) state $x_* = 0.6$

Dynamics : when doing action a_t in state $s_t = (x_t, v_t)$, the next state $s_{t+1} = (x_{t+1}, v_{t+1})$ is

$$\begin{cases} v_{t+1} &= \max\{\min\{v_t + \epsilon_t + 0.001a_t - 0.0025 \cos(3x_t), 0.07\}, -0.07\}, \\ x_{t+1} &= \max\{\min\{x_t + v_t, 0.6\}, -1.2\}. \end{cases}$$

→ for physicists, this may be “continuous space, known dynamics”

Example : Mountain Car



State : $(x, \dot{x}) \in [-1.2; 0.6] \times [-0.07; 0.07]$

Actions : $\mathcal{A} = \{-1, 0, 1\}$:
full speed backwards / do nothing / full speed forward

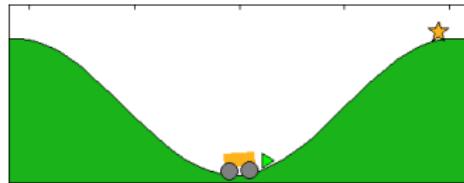
Reward : always -1 except in the terminal (goal) state $x_* = 0.6$

Dynamics : when doing action a_t in state $s_t = (x_t, v_t)$, the next state $s_{t+1} = (x_{t+1}, v_{t+1})$ is

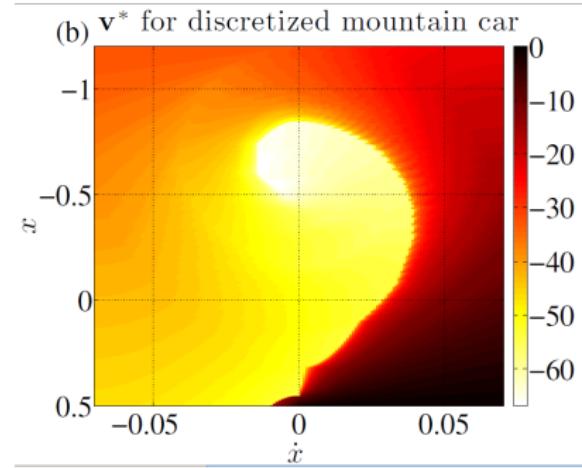
$$\begin{cases} v_{t+1} &= \max\{\min\{v_t + \epsilon_t + 0.001a_t - 0.0025 \cos(3x_t), 0.07\}, -0.07\}, \\ x_{t+1} &= \max\{\min\{x_t + v_t, 0.6\}, -1.2\}. \end{cases}$$

→ for others, this is a “continuous space, unknown dynamics” setting

Example : Mountain Car



The optimal policy is to first climb up the other side :



More “Large space, Unknown Dynamics”

Many concrete problems where RL could be applied fall in this framework

- ▶ micro-grid management
- ▶ self-driving cars
- ▶ autonomous robotics ...

Benchmarks often used by researcher these days are [video games](#) :

- dynamics may be unknown (enemies behavior, random level generation...)
- state-space may be large (e.g., pixels)



Outline

- 1 From Values to Policy Learning**
- 2 Policy Evaluation with Approximation**
- 3 Learning the Optimal Policy : Approximate Dynamic Programming**
- 4 Learning the Optimal Policy : Approximate Q-Learning**

Learning Values or Q-Values

In RL, one often learn values instead of policy directly :

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

Property : $V^*(s) = \max_a Q^*(s, a)$.

From an estimate of V^* to an estimate of Q^*

$$Q \xrightarrow{\text{easy}} V(s) = \max_a Q(s, a)$$

$$V \xrightarrow{\text{possibly harder}} Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot | s, a)} [V(s')]$$

The policy deduced from an estimate V is $\pi = \text{greedy}(V)$

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \left(r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot | s, a)} [V(s')] \right)$$

→ decide when to approximate V^* or Q^*

Learning Values or Q-Values

In RL, one often learn values instead of policy directly :

$$Q^*(s, a) = \max_{\pi} \mathbb{E}^{\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_1 = s, a_1 = a \right]$$

Property : $V^*(s) = \max_a Q^*(s, a)$.

From an estimate of V^* to an estimate of Q^*

$$Q \xrightarrow{\text{easy}} V(s) = \max_a Q(s, a)$$

$$V \xrightarrow{\text{possibly harder}} Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot | s, a)} [V(s')]$$

The policy deduced from an estimate V is $\pi = \text{greedy}(V)$

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \left(r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot | s, a)} [V(s')] \right)$$

→ decide when to approximate V^* or Q^*

Learning Values or Q-Values

In RL, one often learn values instead of policy directly :

$$Q^*(s, a) = \max_{\pi} \mathbb{E}^{\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_1 = s, a_1 = a \right]$$

Property : $V^*(s) = \max_a Q^*(s, a)$.

From an estimate of V^* to an estimate of Q^*

$$Q \xrightarrow{\text{easy}} V(s) = \max_a Q(s, a)$$

$$V \xrightarrow{\text{possibly harder}} Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot | s, a)} [V(s)]$$

The policy deduced from an estimate Q is $\pi = \text{greedy}(Q)$

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$$

→ decide when to approximate V^* or Q^*

From Values to Policies

Question : how does the **approximation error** $\|V - V^*\|$ impact the **performance loss** of the policy deduced from V ?

Proposition

Let V be an approximation of V^* and $\pi = \text{greedy}(V)$.

$$\underbrace{\|V^* - V^\pi\|_\infty}_{\text{performance loss}} \leq \frac{2\gamma}{1-\gamma} \underbrace{\|V^* - V\|_\infty}_{\text{approximation error}} .$$

- ▶ also, $\|V^* - V\|_\infty \leq \|Q^* - Q\|_\infty$ if $V(s) = \max_a Q(s, a)$.

Exercise : Prove it !

Proof

Ingredients :

- ▶ $T^*V^* = V^*$ and $T^\pi V^\pi = V^\pi$
- ▶ both T^* and T^π are γ contractions wrt $\|\cdot\|_\infty$
- ▶ as $\pi = \text{greedy}(V)$, $T^*V = T^\pi V$

$$\begin{aligned}\|V^* - V^\pi\|_\infty &\leq \|T^*V^* - T^\pi V\|_\infty + \|T^\pi V - T^\pi V^\pi\|_\infty \\ &\leq \|T^*V^* - T^*V\|_\infty + \gamma\|V - V^\pi\|_\infty \\ &\leq \gamma\|V^* - V\|_\infty + \gamma(\|V - V^*\|_\infty + \|V^* - V^\pi\|_\infty)\end{aligned}$$

Hence

$$\|V^* - V^\pi\|_\infty \leq \frac{2\gamma}{1-\gamma} \|V^* - V\|_\infty.$$

Value Functions Approximation

Problem : Often \mathcal{S} is too large to store a vector $V \in \mathbb{R}^{\mathcal{S}}$ or a table $Q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ in memory...

Solution : look for estimates V (resp. Q) of V^* (resp. Q^*) in an **approximation space** \mathcal{F}_V (resp. \mathcal{F}_Q)

$$\mathcal{F}_V \subseteq \mathcal{F}(\mathcal{S}, \mathbb{R}) \quad \mathcal{F}_Q \subseteq \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$$

► Parametric approximation :

$$\mathcal{F}_V = \left\{ s \mapsto V_\theta(s) \mid \theta \in \Theta \right\} \quad \mathcal{F}_Q = \left\{ (s, a) \mapsto Q_\theta(s, a) \mid \theta \in \Theta \right\}$$

→ only requires to store a parameter θ (typically in \mathbb{R}^d with $d \ll |\mathcal{S}|$)

Smooth parameterization if $\nabla_\theta V_\theta(s)$ (resp. $\nabla_\theta Q_\theta(s, a)$) can be computed

Value Functions Approximation

Problem : Often \mathcal{S} is too large to store a vector $V \in \mathbb{R}^{\mathcal{S}}$ or a table $Q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ in memory...

Solution : look for estimates V (resp. Q) of V^* (resp. Q^*) in an **approximation space** \mathcal{F}_V (resp. \mathcal{F}_Q)

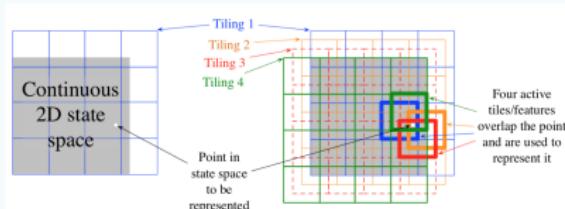
$$\mathcal{F}_V \subseteq \mathcal{F}(\mathcal{S}, \mathbb{R}) \quad \mathcal{F}_Q \subseteq \mathcal{F}(\mathcal{S} \times \mathcal{A}, \mathbb{R})$$

► Non-parametric approximation :

- nearest neighbors
- kernel smoothing

$$V_n(s) = \sum_{t=1}^n v_t \frac{K(x, s_t)}{\sum_{\ell=1}^n K(x, s_\ell)} \quad \text{for some kernel } K$$

- tile coding



Linear function approximation

V is some linear combinations of *basis functions* (or *features*).

$$\mathcal{F}_V = \left\{ s \mapsto V_\theta(s) = \sum_{i=1}^d \theta_i \phi_i(s) \mid \theta \in \mathbb{R}^d \right\}$$

Introducing the **feature vector** of a state s

$$\phi(s) = (\phi_1(s), \dots, \phi_d(s))^\top \in \mathbb{R}^d$$

one can write

$$V_\theta(s) = \theta^\top \phi(s).$$

Remarks :

- ▶ smooth parameterization with $\nabla_\theta V_\theta(s) = \phi(s)$
- ▶ if $\mathcal{S} = \{s_1, \dots, s_S\}$, one recovers the tabular case with $\phi_i(s) = \mathbb{1}(s = s_i)$ for $i = 1, \dots, S$

Linear function approximation

Q is some linear combinations of *basis functions* (or *features*).

$$\mathcal{F}_Q = \left\{ (s, a) \mapsto Q_\theta(s, a) = \sum_{i=1}^d \theta_i \phi_i(s, a) \mid \theta \in \mathbb{R}^d \right\}$$

Introducing the **feature vector** of a state-action pair (s, a)

$$\phi(s, a) = (\phi_1(s, a), \dots, \phi_d(s, a))^\top \in \mathbb{R}^d$$

one can write

$$Q_\theta(s, a) = \theta^\top \phi(s, a).$$

Remarks :

- ▶ smooth parameterization with $\nabla_\theta Q_\theta(s, a) = \phi(s, a)$
- ▶ if $\mathcal{S} = \{s_1, \dots, s_S\}$, $\mathcal{A} = \{a_1, \dots, a_A\}$ one recovers the tabular case with $\phi_{i,j}(s, a) = \mathbb{1}(s = s_i, a = a_j)$ for $i = 1, \dots, S$ and $j = 1, \dots, A$

Examples of features

- ▶ $\mathcal{S} \subseteq \mathbb{R}$: one may use polynomial or Fourier basis
- ▶ $\mathcal{S} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_K$: one may use tensor products of features

$$\phi_{(i_1, \dots, i_K)} \left(\left(s^{(1)}, \dots, s^{(K)} \right) \right) = \prod_{j=1}^K \phi_{i_j} \left(s^{(j)} \right)$$

RBF features

If $\mathcal{S} \subseteq \mathbb{R}^d$, one can use Radial Basic Functions

$$\phi_i(s) = \exp \left(-\eta \|s - s^{(i)}\|^2 \right),$$

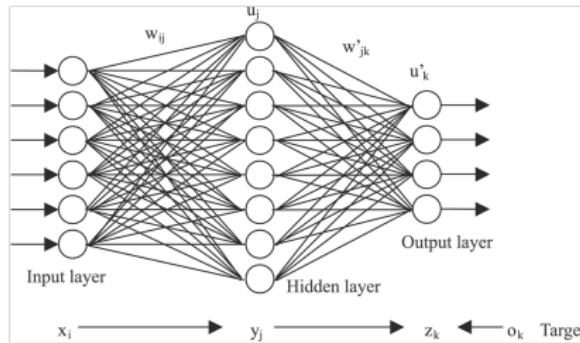
with some scale parameter η and “centers” $s^{(1)}, \dots, s^{(d)}$ (e.g. a uniform covering of \mathcal{S} , or random centers)

Non linear function approximation

Linear function approximation requires to design (meaningful) features, which can be hard...

Modeling V as a neural network can be more powerful :

- ▶ neural networks are known to be universal approximators
- ▶ they “learn features” from the data
- ▶ and $\nabla_{\theta} V_{\theta}(s)$ can still be computed efficiently



Outline

- 1 From Values to Policy Learning**
- 2 Policy Evaluation with Approximation**
- 3 Learning the Optimal Policy : Approximate Dynamic Programming**
- 4 Learning the Optimal Policy : Approximate Q-Learning**

Performance measure

In the tabular case, we proposed algorithms that converge to the **exact** V^π . This is in general hopeless with function approximation.

→ we can instead try to minimize the **Mean Square Error**

Mean Square Value Error

Let ν be some probability measure on the state space \mathcal{S} and $V : \mathcal{S} \rightarrow \mathbb{R}$.

$$\text{MSVE}_\nu(V) = \mathbb{E}_{s \sim \nu} \left[(V^\pi(s) - V(s))^2 \right]$$

Performance measure

In the tabular case, we proposed algorithms that converge to the **exact** V^π . This is in general hopeless with function approximation.

→ we can instead try to minimize the **Mean Square Error**

Mean Square Value Error

Let ν be some probability measure on the state space \mathcal{S} and $V : \mathcal{S} \rightarrow \mathbb{R}$.

$$\text{MSVE}_\nu(V) = \mathbb{E}_{s \sim \nu} \left[(V^\pi(s) - V(s))^2 \right]$$

→ what measure ν do we choose ?

Assumption. Under the policy π , the sequence of visited state $(s_t)_{t \in \mathbb{N}}$ is a Markov chain. We assume that it admits a **stationary distribution** ν .

Performance measure

We proposed before algorithms that converge to the **exact** V^π . This is in general hopeless with function approximation.

→ we can instead try to minimize the **Mean Square Error**

Mean Square Value Error

Let ν be some probability measure on the state space \mathcal{S} and $V : \mathcal{S} \rightarrow \mathbb{R}$.

$$\text{MSVE}_\nu(V) = \mathbb{E}_{s \sim \nu} [(V^\pi(s) - V(s))^2]$$

Remark : defining $\|\cdot\|_\nu$ to be the norm associated to the scalar product

$$\langle f | g \rangle_\nu = \mathbb{E}_{s \sim \nu} [f(s)g(s)],$$

one has

$$\text{MSVE}_\nu(V) = \|V^\pi - V\|_\nu^2$$

Minimizing the MSVE

We consider a smooth parametric representation for V ,
 $\mathcal{F} = \{V_\theta, \theta \in \Theta\}$, for which we can define

$$\text{MSVE}(\theta) = \mathbb{E}_\nu \left[(V^\pi(s) - V_\theta(s))^2 \right]$$

and we aim for $\theta^* = \operatorname{argmin}_{\theta \in \Theta} \text{MSVE}(\theta)$.

Given the smooth parameterization, one can compute

$$\nabla_\theta \text{MSVE}(\theta) = -2\mathbb{E}_\nu [(V^\pi(s) - V_\theta(s)) \nabla_\theta V_\theta(s)]$$

(valid for finite state space, and possibly under some assumption in continuous state spaces)

Gradient descent :

$$\theta_t \leftarrow \theta_{t-1} + \alpha_t \times \mathbb{E}_\nu [(V^\pi(s) - V_{\theta_{t-1}}(s)) \nabla_\theta V_{\theta_{t-1}}(s)]$$

Minimizing the MSVE

We consider a smooth parametric representation for V ,
 $\mathcal{F} = \{V_\theta, \theta \in \Theta\}$, for which we can define

$$\text{MSVE}(\theta) = \mathbb{E}_\nu \left[(V^\pi(s) - V_\theta(s))^2 \right]$$

and we aim for $\theta^* = \operatorname{argmin}_{\theta \in \Theta} \text{MSVE}(\theta)$.

Given the smooth parameterization, one can compute

$$\nabla_\theta \text{MSVE}(\theta) = -2\mathbb{E}_\nu [(V^\pi(s) - V_\theta(s)) \nabla_\theta V_\theta(s)]$$

(valid for finite state space, and possibly under some assumption in continuous state spaces)

Stochastic gradient descent :

$$\theta_t \leftarrow \theta_{t-1} + \alpha_t \times (V^\pi(s_t) - V_{\theta_{t-1}}(s_t)) \nabla_\theta V_{\theta_{t-1}}(s_t)$$

(for large t , s_t is approximately distributed under ν)

Minimizing the MSVE

We consider a smooth parametric representation for V ,
 $\mathcal{F} = \{V_\theta, \theta \in \Theta\}$, for which we can define

$$\text{MSVE}(\theta) = \mathbb{E}_\nu \left[(V^\pi(s) - V_\theta(s))^2 \right]$$

and we aim for $\theta^* = \operatorname{argmin}_{\theta \in \Theta} \text{MSVE}(\theta)$.

Given the smooth parameterization, one can compute

$$\nabla_\theta \text{MSVE}(\theta) = -2\mathbb{E}_\nu [(V^\pi(s) - V_\theta(s)) \nabla_\theta V_\theta(s)]$$

(valid for finite state space, and possibly under some assumption in continuous state spaces)

Stochastic gradient descent :

$$\theta_t \leftarrow \theta_{t-1} + \alpha_t \times (V^\pi(s_t) - V_{\theta_{t-1}}(s_t)) \nabla_\theta V_{\theta_{t-1}}(s_t)$$

(for large t , s_t is approximately distributed under ν)

→ problem : $V^\pi(s_t)$ is unknown...

A semi-gradient approach

Idea : in the stochastic gradient descent update

$$\theta_t \leftarrow \theta_{t-1} + \alpha_t \times (V^\pi(s_t) - V_{\theta_{t-1}}(s_t)) \nabla_\theta V_{\theta_{t-1}}(s_t)$$

replace $V^\pi(s_t)$ by either

- ▶ a Monte-Carlo estimate (TD(1))
- ▶ a “Bootstrap” estimate (TD(0))

TD(0) with smooth function approximation

The TD(0) **semi-gradient** update is

$$\theta_t \leftarrow \theta_{t-1} + \alpha_t \times (r_t + \gamma V_{\theta_{t-1}}(s_{t+1}) - V_{\theta_{t-1}}(s_t)) \nabla_\theta V_{\theta_{t-1}}(s_t)$$

⚠ this is *not* a stochastic gradient update, hence the terminology

- stepsize tuning : decaying not too fast (Robbins-Monro style)
- very few convergence guarantees besides the **linear case**...

TD(0) with linear function approximation

We assume $V_\theta(s) = \theta^\top \phi(s)$ with the feature vector

$$\phi(s) = (\phi_1(s), \dots, \phi_d(s))^\top \in \mathbb{R}^d.$$

Then $\nabla_\theta V_\theta(s) = \phi(s)$ and the algorithm becomes

TD(0) with linear function approximation

Along a trajectory following π , after observing (s_t, r_t, s_{t+1}) update

$$\theta_t = \theta_{t-1} + \alpha_t (r_t + \gamma \theta_{t-1}^\top \phi(s_{t+1}) - \theta_{t-1}^\top \phi(s_t)) \phi(s_t).$$

Using the notation $\phi_t = \phi(s_t)$, one has

$$\theta_t = \theta_{t-1} + \alpha_t (r_t \phi_t - \phi_t (\phi_t - \gamma \phi_{t+1})^\top \theta_{t-1}).$$

Convergence properties

Theorem

Under the following assumptions :

- ① the Markov chain $(s_t)_{t \in \mathbb{N}}$ admits a stationary distribution ν
- ② the state space is finite and the vectors $\phi_i = (\phi_i(s))_{s \in \mathcal{S}} \in \mathbb{R}^S$ are linearly independent
- ③ the step-sizes satisfy the Robbins-Monro conditions, i.e.

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha_t < \infty$$

then the parameter θ_t converges almost surely to some value θ_{TD} s.t.

$$V_{\theta_{\text{TD}}} = \underbrace{\Pi_{\mathcal{F}, \nu} T^\pi}_{\substack{\text{projected} \\ \text{Bellman operator}}} V_{\theta_{\text{TD}}}$$

$$\Pi_{\mathcal{F}, \nu} T^\pi(V) = \operatorname{argmin}_{f \in \mathcal{F}} \|T^\pi(V) - f\|_\nu$$

Computing the fixed point

According to the theorem, TD(0) converges to the solution to

$$V_{\theta_{\text{TD}}} = \Pi_{\mathcal{F}, \nu} T^\pi V_{\theta_{\text{TD}}}$$

Proposition

The vector θ_{TD} can be obtained as a solution to the linear system

$$A^\pi \theta_{\text{TD}} = b^\pi,$$

where

$$\begin{aligned} A_{i,j}^\pi &= \langle \phi_i | \phi_j - \gamma P^\pi \phi_j \rangle_\nu \\ b_i^\pi &= \langle r^\pi | \phi_i \rangle \end{aligned}$$

Computing the fixed point

According to the theorem, TD(0) converges to the solution to

$$V_{\theta_{\text{TD}}} = \Pi_{\mathcal{F}, \nu} T^\pi V_{\theta_{\text{TD}}}$$

Proposition

The vector θ_{TD} can be obtained as a solution to the linear system

$$A^\pi \theta_{\text{TD}} = b^\pi,$$

where

$$\begin{aligned} A^\pi &= \mathbb{E}_{\substack{s' \sim \nu \\ s' \sim p(\cdot | s, \pi(s))}} \left[\phi(s) (\phi(s) - \gamma \phi(s'))^\top \right] \in \mathbb{R}^{d \times d} \\ b^\pi &= \mathbb{E}_{s \sim \nu} [r(s, \pi(s)) \phi(s)] \in \mathbb{R}^d \end{aligned}$$

Why does TD(0) converge to θ_{TD} ?

(heuristic argument in [Sutton and Barto, 2018])

Recall the TD(0) update :

$$\begin{aligned}\theta_t &= \theta_{t-1} + \alpha_t (r_t \phi(s_t) - \phi(s_t)(\phi(s_t) - \gamma \phi(s_{t+1})))^\top \theta_{t-1}) \\ &= \theta_{t-1} + \alpha_t (b_t - A_t \theta_{t-1}),\end{aligned}$$

where we introduce

$$\begin{aligned}A_t &= \phi(s_t)(\phi(s_t) - \gamma \phi(s_{t+1}))^\top \in \mathbb{R}^{d \times d} \\ b_t &= r_t \phi(s_t) \in \mathbb{R}^d\end{aligned}$$

Why does TD(0) converge to θ_{TD} ?

(heuristic argument in [Sutton and Barto, 2018])

Recall the TD(0) update :

$$\begin{aligned}\theta_t &= \theta_{t-1} + \alpha_t (r_t \phi(s_t) - \phi(s_t)(\phi(s_t) - \gamma \phi(s_{t+1})))^\top \theta_{t-1}) \\ &= \theta_{t-1} + \alpha_t (b_t - A_t \theta_{t-1}),\end{aligned}$$

where we introduce

$$\begin{aligned}A_t &\simeq \mathbb{E}_{s_t \sim \nu} [\phi(s_t)(\phi(s_t) - \gamma \phi(s_{t+1}))^\top] = A^\pi \\ b_t &\simeq \mathbb{E}_{s_t \sim \nu} [r_t \phi(s_t)] = b^\pi\end{aligned}$$

when t is large as s_t is approximately drawn under ν .

Why does TD(0) converge to θ_{TD} ?

(heuristic argument in [Sutton and Barto, 2018])

Recall the TD(0) update :

$$\begin{aligned}\theta_t &= \theta_{t-1} + \alpha_t (r_t \phi(s_t) - \phi(s_t)(\phi(s_t) - \gamma \phi(s_{t+1})))^\top \theta_{t-1}) \\ &= \theta_{t-1} + \alpha_t (b_t - A_t \theta_{t-1}),\end{aligned}$$

where we introduce

$$\begin{aligned}A_t &\simeq \mathbb{E}_{s_t \sim \nu} [\phi(s_t)(\phi(s_t) - \gamma \phi(s_{t+1}))^\top] = A^\pi \\ b_t &\simeq \mathbb{E}_{s_t \sim \nu} [r_t \phi(s_t)] = b^\pi\end{aligned}$$

when t is large as s_t is approximately drawn under ν .

Approximate recursion :

$$\theta_t = \theta_{t-1} + \alpha (b^\pi - A^\pi \theta_{t-1})$$

If it converges, the convergence is towards a fixed point, satisfying

$$b^\pi - A^\pi \theta = 0$$

Least Square Temporal Difference

Idea : Now that we know towards what TD(0) converges, is there a way to get there faster ?

$$A^\pi \theta_{\text{TD}} = b^\pi,$$

where

$$\begin{aligned} A^\pi &= \mathbb{E}_{\substack{s \sim \nu \\ s' \sim p(\cdot | s, \pi(s))}} \left[\phi(s) (\phi(s) - \gamma \phi(s'))^\top \right] \in \mathbb{R}^{d \times d} \\ b^\pi &= \mathbb{E}_{s \sim \nu} [r(s, \pi(s)) \phi(s)] \in \mathbb{R}^d \end{aligned}$$

► use estimation :

$$\hat{A}_n = \frac{1}{n} \sum_{t=1}^n \phi(s_t) (\phi(s_t) - \gamma \phi(s_{t+1}))^\top \quad \text{and} \quad \hat{b}_n = \frac{1}{n} \sum_{t=1}^n r_t \phi(s_t)$$

If \hat{A}_n is invertible, $\hat{\theta}_n = \hat{A}_n^{-1} \hat{b}_n$.

An online implementation of LSTD

We need to compute

$$\hat{\theta}_n = \hat{A}_n^{-1} \hat{b}_n$$

where

$$\hat{A}_n = \sum_{t=1}^n \phi(s_t) (\phi(s_t) - \gamma \phi(s_{t+1}))^\top \quad \text{and} \quad \hat{b}_n = \sum_{t=1}^n r_t \phi(s_t).$$

- requires to invert a $d \times d$ matrix at every round...
(much more costly than the TD(0) update!)

More efficient : update the inverse online !

Sherman-Morrison formula

For any matrix $B \in \mathbb{R}^{d \times d}$ and vectors $u, v \in \mathbb{R}^d$,

$$(B + uv^\top)^{-1} = B^{-1} - \frac{B^{-1}uv^\top B^{-1}}{1 + v^\top B^{-1}u}$$

LSTD update versus TD(0) update

Letting $\phi_t = \phi(s_t)$, both update also rely on temporal differences

$$\delta_t(\theta) = r_t + \gamma\phi_{t+1}^\top\theta - \phi_t^\top\theta$$

Recursive LSTD

$$\begin{aligned} C_n &= C_{n-1} - \frac{C_{n-1}\phi_n(\phi_n - \gamma\phi_{n+1})^\top C_{n-1}}{1 + (\phi_n - \gamma\phi_{n+1})^\top C_{n-1}\phi_n} \\ \theta_n &= \theta_{n-1} + \frac{C_{n-1}}{1 + (\phi_n - \gamma\phi_{n+1})^\top C_{n-1}\phi_n} \delta_n(\theta_{n-1})\phi_n \end{aligned}$$

TD(0)

$$\theta_n = \theta_{n-1} + \alpha_n \delta_n(\theta_{n-1})\phi_n$$

Complexity : $O(d^2)$ versus $O(1)$ but LSTD converges faster

Wait... How good is the TD solution ?

We presented two algorithms which converge to the value function

$$V_{\text{TD}}(s) = \theta_{\text{TD}}^\top \phi(s)$$

such that V_{TD} is a fixed point to $\Pi_{\mathcal{F}, \nu} T^\pi$ (when it exists).

→ Is it at all close to our target V^π ?

Proposition

If ν is the stationary distribution of the sequence of states

- ▶ $\Pi_{\mathcal{F}, \nu} T^\pi$ is a γ contraction with respect to $\|\cdot\|_\nu$ and admits therefore a unique fixed point, V_{TD}
- ▶ The TD solution satisfies

$$\|V^\pi - V_{\text{TD}}\|_\nu \leq \frac{1}{\sqrt{1 - \gamma^2}} \inf_{V \in \mathcal{F}} \|V^\pi - V\|_\nu$$

Answer : not too far from the best possible approximation (wrt to $\|\cdot\|_\nu$)

Outline

- 1 From Values to Policy Learning
- 2 Policy Evaluation with Approximation
- 3 Learning the Optimal Policy : Approximate Dynamic Programming
- 4 Learning the Optimal Policy : Approximate Q-Learning

Reminder : Policy Iteration

- ➊ Let π_0 be *any* stationary policy
- ➋ At each iteration $k = 1, 2, \dots, K$
 - **Policy evaluation** : given π_{k-1} , compute $V_k = V^{\pi_{k-1}}$.
 - **Policy improvement** : compute the *greedy* policy

$$\pi_k(s) \in \operatorname{argmax}_{a \in \mathcal{A}} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot | s, a)}[V_{k-1}(s')]].$$

- ➌ Return the last policy π_K

Reminder : Policy Iteration

- ➊ Let π_0 be *any* stationary policy
- ➋ At each iteration $k = 1, 2, \dots, K$

- Policy evaluation : given π_{k-1} , compute $V_k = V^{\pi_{k-1}}$.
- Policy improvement : compute the *greedy* policy

$$\pi_k(s) \in \operatorname{argmax}_{a \in \mathcal{A}} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot | s, a)}[V_{k-1}(s')]].$$

- ➌ Return the last policy π_K

- ▶ **Problem** : we saw how to approximately perform policy evaluation, how about policy improvement ?

Reminder : Policy Iteration

- ➊ Let π_0 be *any* stationary policy
- ➋ At each iteration $k = 1, 2, \dots, K$
 - **Policy evaluation** : given π_{k-1} , compute $V_k = V^{\pi_{k-1}}$.
 - **Policy improvement** : compute the *greedy* policy

$$\pi_k(s) \in \operatorname{argmax}_{a \in \mathcal{A}} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot | s, a)}[V_{k-1}(s')]].$$

- ➌ Return the last policy π_K

- ▶ **Problem** : we saw how to approximately perform policy evaluation,
how about policy improvement?
- work with Q-values directly to make policy improvement easy !

LSTD-Q

LSTD-Q : a variant of LSTD aimed at estimating directly Q^π

$$Q_\theta(s, a) = \theta^\top \phi(s, a)$$

The solution to

$$Q_\theta = \Pi_{\mathcal{F}, \nu} T^\pi Q_\theta$$

can similarly be approximated by solving a linear system.

$$\begin{cases} A_n &= A_{n-1} + \phi(s_n, a_n)(\phi(s_n, a_n) - \gamma \phi(s_{n+1}, \pi(s_{n+1})))^\top \\ b_n &= b_{n-1} + \phi(s_n, a_n)r_n \end{cases}$$

$$\theta_n^{\text{LSTD-Q}} = A_n^{-1} b_n$$

- The resulting algorithm is **Least-Squares Policy Iteration** (LSPI)
[Lagoudakis and Parr, 2003]

Reminder : Value Iteration

- ❶ Let Q_0 be *any* action-value function
- ❷ At each iteration $k = 1, 2, \dots, K$

$$\begin{aligned} Q_k(s, a) &= T^* Q_{k-1}(s, a) \\ &= r(s, a) + \mathbb{E}_{s' \sim p(\cdot | s, a)} \left[\max_{a' \in \mathcal{A}} Q_{k-1}(s', a') \right] \end{aligned}$$

- ❸ Return the greedy policy

$$\pi_K(s) \in \operatorname{argmax}_{a \in \mathcal{A}} Q_K(s, a).$$

Reminder : Value Iteration

- ➊ Let Q_0 be *any* action-value function
- ➋ At each iteration $k = 1, 2, \dots, K$

$$\begin{aligned} Q_k(s, a) &= T^* Q_{k-1}(s, a) \\ &= r(s, a) + \mathbb{E}_{s' \sim p(\cdot | s, a)} \left[\max_{a' \in \mathcal{A}} Q_{k-1}(s', a') \right] \end{aligned}$$

- ➌ Return the greedy policy

$$\pi_K(s) \in \operatorname{argmax}_{a \in \mathcal{A}} Q_K(s, a).$$

- **Problem** : how can we approximate $T^* Q_k$?
- **Problem** : does value iteration still work with such an approximation ?

Fitted-Q Iteration

Input : number of iterations K , number of samples per iteration n ,
Initial function $Q_0 \in \mathcal{F}$, sampling distribution ρ ,
Approximation space \mathcal{F} , loss function ℓ

```
1 for  $k = 1, \dots, K$  do
2   Draw  $n$  samples  $(s_i, a_i) \sim \rho$ 
3   Perform  $n$  transitions  $r_i, s'_i = \text{step}(s_i, a_i)$ 
4   Compute the targets  $y_i = r_i + \gamma \max_a Q_{k-1}(s'_i, a)$ 
5   From the training dataset  $\mathcal{D}_k = \{(s_i, a_i), y_i\}_{1 \leq i \leq n}$ , solve the
6   empirical risk minimization problem :
```

$$f \in \operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(s_i, a_i))$$

```
7   Set  $Q_k = f$  (with clipping if  $f(s, a) \notin [-\frac{R_{\max}}{1-\gamma}; \frac{R_{\max}}{1-\gamma}]$ ).
```

```
8 end
```

Return: $\pi = \text{greedy}(Q_K)$

- ▶ ERM can be replaced by other possibly non-parameteric regression techniques (decision trees, k -nn, ...)

Linear Fitted Q-Iteration

Input : number of iterations K , number of samples per iteration n ,
Initial function $Q_0 \in \mathcal{F}$, sampling distribution ρ ,
Approximation space \mathcal{F} , loss function ℓ

- 1 **for** $k = 1, \dots, K$ **do**
- 2 Draw n samples $(s_i, a_i) \sim \rho$
- 3 Perform n transitions $r_i, s'_i = \text{step}(s_i, a_i)$
- 4 Compute the targets $y_i = r_i + \gamma \max_a Q_{k-1}(s'_i, a)$
- 5 From the training dataset $\mathcal{D}_k = \{(s_i, a_i), y_i\}_{1 \leq i \leq n}\}$, solve the
least squares problem :
- 6

$$\theta_k \in \operatorname{argmin}_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (y_i - \theta^\top \phi(s_i, a_i))^2$$

- 7 Set $Q_k(s, a) = \theta_k^\top \phi(s, a)$ (with clipping).
- 8 **end**

Return: $\pi = \text{greedy}(Q_K)$

Linear Fitted-Q : Sampling

- ❶ Draw n samples $(s_i, a_i) \stackrel{\text{i.i.d}}{\sim} \rho$
- ❷ Perform a transition for each of them :
 $s'_i \sim p(\cdot | s_i, a_i)$ and $r_i \sim \nu_{(s_i, a_i)}$

Linear Fitted-Q : Sampling

- ➊ Draw n samples $(s_i, a_i) \stackrel{\text{i.i.d}}{\sim} \rho$
- ➋ Perform a transition for each of them :
 $s'_i \sim p(\cdot | s_i, a_i)$ and $r_i \sim \nu_{(s_i, a_i)}$

- In practice sampling can be done **once** before running the algorithm (or a database of transitions can be used)
- The sampling distribution ρ should cover the state-action space in all *relevant regions*
- The algorithm requires call to a **simulator** which can simulate *independent transitions* from anywhere in the state-action space

Linear Fitted-Q : Building the training set

- ③ Compute $y_i = r_i + \gamma \max_a Q_{k-1}(s'_i, a)$
- ④ Build training set $\mathcal{D}_k = \{(s_i, a_i), y_i\}_{1 \leq i \leq n}$

Linear Fitted-Q : Building the training set

- ❸ Compute $y_i = r_i + \gamma \max_a Q_{k-1}(s'_i, a)$
- ❹ Build training set $\mathcal{D}_k = \{(s_i, a_i), y_i\}_{1 \leq i \leq n}$

→ Each sample y_i is an unbiased estimate of $T^*Q_{k-1}(s_i, a_i)$:

$$\begin{aligned}\mathbb{E}[y_i | s_i, a_i, Q_{k-1}] &= \mathbb{E}[r_i + \gamma \max_{a'} Q_{k-1}(s'_i, a') | s_i, a_i, Q_{k-1}] \\ &= r(s_i, a_i) + \gamma \mathbb{E}_{s' \sim p(\cdot | s_i, a_i)} [\max_{a'} Q_{k-1}(s', a')] \\ &= T^*Q_{k-1}(s_i, a_i)\end{aligned}$$

- The problem “reduces” to standard regression
- A new regression problem at each iteration :
new function to fit T^*Q_{k-1} + new training set \mathcal{D}_k

Linear Fitted-Q : The regression problem

- ➅ Solve the least squares problem

$$\theta_k \in \operatorname{argmin}_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (y_i - \theta^\top \phi(s_i, a_i))^2$$

Linear Fitted-Q : The regression problem

- ➅ Solve the least squares problem

$$\theta_k \in \operatorname{argmin}_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (y_i - \theta^\top \phi(s_i, a_i))^2$$

→ standard linear regression problem with design matrix and targets

$$X = \begin{pmatrix} \phi(s_1, a_1)^\top \\ \phi(s_2, a_2)^\top \\ \dots \\ \phi(s_n, a_n)^\top \end{pmatrix} \in \mathbb{R}^{n \times d} \quad \text{and} \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} \in \mathbb{R}^d$$

whose solution is

$$\theta_k = (X^\top X)^{-1} X^\top Y.$$

Linear Fitted-Q : Error bound

Theorem

Linear FQI with a space \mathcal{F} of d features, with n samples drawn from ρ at each iteration, returns a policy π_K after K iterations which satisfies, w.p. larger than $1 - \delta$,

$$\begin{aligned} \|Q^* - Q^{\pi_K}\|_\mu &\leq \frac{2\gamma}{(1-\gamma)^2} C_{\mu,\rho} \left[\sup_{g \in \mathcal{F}} \inf_{f \in \mathcal{F}} \|T^* g - f\|_\rho \right. \\ &\quad \left. + O\left(\sqrt{\frac{d \log(\textcolor{teal}{n}/\delta)}{\omega \textcolor{teal}{n}}}\right) \right] \\ &\quad + O\left(\frac{\gamma^K}{(1-\gamma)^2}\right). \end{aligned}$$

see, e.g. [Munos and Szepesvári, 2008]

Outline

- 1 From Values to Policy Learning**
- 2 Policy Evaluation with Approximation**
- 3 Learning the Optimal Policy : Approximate Dynamic Programming**
- 4 Learning the Optimal Policy : Approximate Q-Learning**

A semi-gradient extension of Q-Learning

Let's try to find θ minimizing

$$\begin{aligned}\text{MSE}(\theta) &= \mathbb{E}_\nu \left[(Q^*(s, a) - Q_\theta(s, a))^2 \right] \\ \nabla_\theta \text{MSE}(\theta) &= -2\mathbb{E}_\nu [(Q^*(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)]\end{aligned}$$

A semi-gradient extension of Q-Learning

Let's try to find θ minimizing

$$\begin{aligned}\text{MSE}(\theta) &= \mathbb{E}_\nu \left[(Q^*(s, a) - Q_\theta(s, a))^2 \right] \\ \nabla_\theta \text{MSE}(\theta) &= -2\mathbb{E}_\nu [(Q^*(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)]\end{aligned}$$

→ gradient descent :

$$\theta \leftarrow \theta + \alpha \mathbb{E}_\nu [(Q^*(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)]$$

A semi-gradient extension of Q-Learning

Let's try to find θ minimizing

$$\begin{aligned}\text{MSE}(\theta) &= \mathbb{E}_\nu \left[(Q^*(s, a) - Q_\theta(s, a))^2 \right] \\ \nabla_\theta \text{MSE}(\theta) &= -2\mathbb{E}_\nu [(Q^*(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)]\end{aligned}$$

→ gradient descent :

$$\theta \leftarrow \theta + \alpha \mathbb{E}_\nu [(Q^*(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)]$$

→ stochastic gradient descent : if $(s_t, a_t) \sim \nu$,

$$\theta \leftarrow \theta + \alpha (Q^*(s_t, a_t) - Q_\theta(s_t, a_t)) \nabla_\theta Q_\theta(s_t, a_t)$$

A semi-gradient extension of Q-Learning

Let's try to find θ minimizing

$$\begin{aligned}\text{MSE}(\theta) &= \mathbb{E}_\nu \left[(Q^*(s, a) - Q_\theta(s, a))^2 \right] \\ \nabla_\theta \text{MSE}(\theta) &= -2\mathbb{E}_\nu [(Q^*(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)]\end{aligned}$$

→ gradient descent :

$$\theta \leftarrow \theta + \alpha \mathbb{E}_\nu [(Q^*(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)]$$

→ stochastic gradient descent : if $(s_t, a_t) \sim \nu$,

$$\theta \leftarrow \theta + \alpha (Q^*(s_t, a_t) - Q_\theta(s_t, a_t)) \nabla_\theta Q_\theta(s_t, a_t)$$

→ bootstrapping : given a transition (s_t, a_t, r_t, s_{t+1}) ,

$$\theta \leftarrow \theta + \alpha \left(r_t + \gamma \max_b Q_\theta(s_{t+1}, b) - Q_\theta(s_t, a_t) \right) \nabla_\theta Q_\theta(s_t, a_t)$$

A semi-gradient extension of Q-Learning

Let's try to find θ minimizing

$$\begin{aligned}\text{MSE}(\theta) &= \mathbb{E}_\nu \left[(Q^*(s, a) - Q_\theta(s, a))^2 \right] \\ \nabla_\theta \text{MSE}(\theta) &= -2\mathbb{E}_\nu [(Q^*(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a)]\end{aligned}$$

Q-Learning update with function approximation

Given a Q-value $Q_\theta(s, a)$, this **semi-gradient** update is

$$\begin{cases} \delta_t &= r_t + \gamma \max_b Q_{\theta_{t-1}}(s_{t+1}, b) - Q_{\theta_{t-1}}(s_t, a_t) \\ \theta_t &= \theta_{t-1} + \alpha_t \delta_t \nabla_\theta Q_{\theta_{t-1}}(s_t, a_t) \end{cases}$$

→ one recovers Q-Learning in the tabular case

Negative results

	TD(0)	LSPI	Fitted-Q	Q-Learning
Linear functions	✓	✓	✓	✗
Non-linear functions	✗	✗	(✓)	✗

- ▶ TD(0) is known to diverge with non-linear function approximation
- ▶ Q-Learning can already diverge with linear function approximation...

(see examples in [Sutton and Barto, 2018])

Tricks to make Q-Learning work

Q-Learning update with function approximation

$$\begin{cases} \delta_t &= r_t + \gamma \max_b Q_{\theta_{t-1}}(s_{t+1}, b) - Q_{\theta_{t-1}}(s_t, a_t) \\ \theta_t &= \theta_{t-1} + \alpha_t \delta_t \nabla_{\theta} Q_{\theta_{t-1}}(s_t, a_t) \end{cases}$$

Alternative view : in each step t , perform one SGD step on

$$L(\theta) = \mathbb{E}_{\substack{(s,a) \sim \rho \\ (r,s') \sim \text{step}(s,a)}} \left[\left(r + \gamma \max_b Q_{\theta_{t-1}}(s', b) - Q_{\theta}(s, a) \right)^2 \right]$$

where ρ is the current behavior policy.

Tricks to make Q-Learning work

Q-Learning update with function approximation

$$\begin{cases} \delta_t &= r_t + \gamma \max_b Q_{\theta_{t-1}}(s_{t+1}, b) - Q_{\theta_{t-1}}(s_t, a_t) \\ \theta_t &= \theta_{t-1} + \alpha_t \delta_t \nabla_{\theta} Q_{\theta_{t-1}}(s_t, a_t) \end{cases}$$

Alternative view : in each step t , perform one SGD step on

$$L(\theta) = \mathbb{E}_{\substack{(s,a) \sim \rho \\ (r,s') \sim \text{step}(s,a)}} \left[\left(r + \gamma \max_b Q_{\theta_{t-1}}(s', b) - Q_{\theta}(s, a) \right)^2 \right]$$

where ρ is the current behavior policy.

Three tricks : (e.g. [Mnih et al., 2015, Hessel et al., 2018])

- experience replay : rely on past transitions instead of the current one
- mini-batches : rely on more than one transition
- two learning scales : do not update the target network in every round

Tricks to make Q-Learning work

Q-Learning update with function approximation

$$\begin{cases} \delta_t &= r_t + \gamma \max_b Q_{\theta_{t-1}}(s_{t+1}, b) - Q_{\theta_{t-1}}(s_t, a_t) \\ \theta_t &= \theta_{t-1} + \alpha_t \delta_t \nabla_{\theta} Q_{\theta_{t-1}}(s_t, a_t) \end{cases}$$

Alternative view : in each step t , perform one SGD step on

$$L(\theta) = \mathbb{E}_{\substack{(s,a) \sim \rho \\ (r,s') \sim \text{step}(s,a)}} \left[\left(r + \gamma \max_b Q_{\theta_{t-1}}(s', b) - Q_{\theta}(s, a) \right)^2 \right]$$

where ρ is the current behavior policy.

Three tricks : (e.g. [Mnih et al., 2015, Hessel et al., 2018])

- **experience replay** : rely on past transitions instead of the current one
- **mini-batches** : rely on more than one transition
- **two learning scales** : do not update the target network in every round

Tricks to make Q-Learning work

Q-Learning update with function approximation

$$\begin{cases} \delta_t &= r_t + \gamma \max_b Q_{\theta_{t-1}}(s_{t+1}, b) - Q_{\theta_{t-1}}(s_t, a_t) \\ \theta_t &= \theta_{t-1} + \alpha_t \delta_t \nabla_{\theta} Q_{\theta_{t-1}}(s_t, a_t) \end{cases}$$

Alternative view : in each step t , perform one SGD step on

$$L(\theta) = \mathbb{E}_{\substack{(s,a) \sim \rho \\ (r,s') \sim \text{step}(s,a)}} \left[\left(r + \gamma \max_b Q_{\theta_{t-1}}(s', b) - Q_{\theta}(s, a) \right)^2 \right]$$

where ρ is the current behavior policy.

Three tricks : (e.g. [Mnih et al., 2015, Hessel et al., 2018])

- experience replay : rely on past transitions instead of the current one
- mini-batches : rely on more than one transition
- two learning scales : do not update the target network in every round

Tricks to make Q-Learning work

Q-Learning update with function approximation

$$\begin{cases} \delta_t &= r_t + \gamma \max_b Q_{\theta_{t-1}}(s_{t+1}, b) - Q_{\theta_{t-1}}(s_t, a_t) \\ \theta_t &= \theta_{t-1} + \alpha_t \delta_t \nabla_{\theta} Q_{\theta_{t-1}}(s_t, a_t) \end{cases}$$

Alternative view : in each step t , perform one SGD step on

$$L(\theta) = \mathbb{E}_{\substack{(s,a) \sim \rho \\ (r,s') \sim \text{step}(s,a)}} \left[\left(r + \gamma \max_b Q_{\theta_{t-1}}(s', b) - Q_{\theta}(s, a) \right)^2 \right]$$

where ρ is the current behavior policy.

Three tricks : (e.g. [Mnih et al., 2015, Hessel et al., 2018])

- experience replay : rely on past transitions instead of the current one
- mini-batches : rely on more than one transition
- two learning scales : do not update the target network in every round

Deep Q Networks

Input : number of iterations T ,
minibatch size B , update frequency for the target network N ,
exploration sequence (ε_t) , stepsize (α_t)
Initialize : replay buffer $\mathcal{D} \leftarrow \{\}$, first state s_1 ,
online network parameter θ , target network parameter $\theta_- \leftarrow \theta$

1 **for** $t = 1, \dots, T$ **do**

2 $a_t = \operatorname{argmax}_a Q_\theta(s_t, a)$ w.p. $1 - \varepsilon_t$, random action w.p. ε_t

3 Perform transition $(r_t, s_{t+1}) = \text{step}(s_t, a_t)$

4 Add transition to the replay buffer $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$

5 Draw a minibatch \mathcal{B} of size B uniformly from \mathcal{D}

6 Perform one step of online optimization on the loss function

$$L(\theta) = \sum_{(s, a, r, s') \in \mathcal{B}} \left(r + \gamma \max_b Q_{\theta^-}(s', b) - Q_\theta(s, a) \right)^2$$

8 e.g. $\theta \leftarrow \theta - \alpha_t \nabla_\theta L(\theta)$

9 every N time steps, $\theta^- \leftarrow \theta$

10 **end**

Return: Q_θ

Results on Atari Games

DQN was proposed in combination with

- ▶ a well chosen pre-processing of the state
- ▶ an optimized architecture for the Deep Neural Network used for the approximator

that reaches super-human level performance on Atari games.



[video]

Summary

In this class, we mostly saw how to scale up reinforcement learning with
Value-based methods :

- ▶ Fitted-Q Iteration
- ▶ Deep Q Networks

In the sequel, we will see :

- ▶ **Policy-based** methods (based on direct search over a policy space)
- ▶ **Actor-critic** methods (using both a policy and a value),
whose performance can also be “boosted” with Deep Learning.

We will also discuss the **exploration** issue : can we go beyond ε -greedy ?
(starting with very simple MDPs : multi-armed bandits)

-  Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018).
Rainbow : Combining improvements in deep reinforcement learning.
In *Thirty-second AAAI conference on artificial intelligence*.
-  Lagoudakis, M. G. and Parr, R. (2003).
Least-squares policy iteration.
The Journal of Machine Learning Research, 4 :1107–1149.
-  Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015).
Human-level control through deep reinforcement learning.
nature, 518(7540) :529–533.
-  Munos, R. and Szepesvári, C. (2008).
Finite-time bounds for fitted value iteration.
Journal of Machine Learning Research, 9(5).
-  Sutton, R. S. and Barto, A. G. (2018).
Reinforcement learning : An introduction.
MIT press.



Tsitsiklis, J. and Van Roy, B. (1996).

Analysis of temporal-difference learning with function approximation.

Advances in neural information processing systems, 9.