SD = REINFORCEMENT LEARNING

" Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal " ≡ " learning from interactions to achieve goals "

[R. S. Sutton & A. G. Barto. Reinforcement Learning = An Introduction.]

↳ In in Reinforcement Learning (RL) problem, there is no supervisor, only a reward (≠ from Supervised Learning) + data is sequential : time really matters.

The learning agent's actions influence the subsequent data it receives → he should be able to sense the state of his environment, and must be able to take actions that affect the state. The agent must also have goals related to the state of the environment. The mathematical model that captures those aspects is known as Markov Decision Process. (MDP).

Ex: (i) Defeat the world champion at Backgammon / Go
(ii) Make a robot walk
(iii) Manage an investment portfolio.
(iv) Adaptive Controller adjusts parameters of a petroleum refinery's operation in real time.

All these examples involve interaction between an agent & its environment; where the agent seeks to achieve a goal despite uncertainty about its environment (next move at 'go', the robot's next location, where to invest (which assets, how much), levels of the reservoirs). The agent's actions affect the options available at later times (→ feedback is delayed, not instantaneous). Correct choices require taking into account delayed consequences of actions → planning required; think ahead !

Elements of RL

- A policy = how the agent picks its actions.
  = a mapping from perceived states of the environment to actions to be taken.
- A reward signal = how well an agent is doing at time t.
  The agent's goal is to maximize the cumulative reward.
  ≡ immediate reward
  ↳ primary basis for altering the policy.
- A value function = captures what is good in the long run (as opposed to the reward signal) : how good is each state and / or actions ?
  The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- A model of the environment = the agent's representation of the environment.
  ↳ Optional. Methods in RF that use a model are known as model-based approaches (first learn the environment model, and then use that to learn a policy), as opposed to model-free approaches (learning an action policy directly).

I- FINITE MARKOV DECISION PROCESSES.

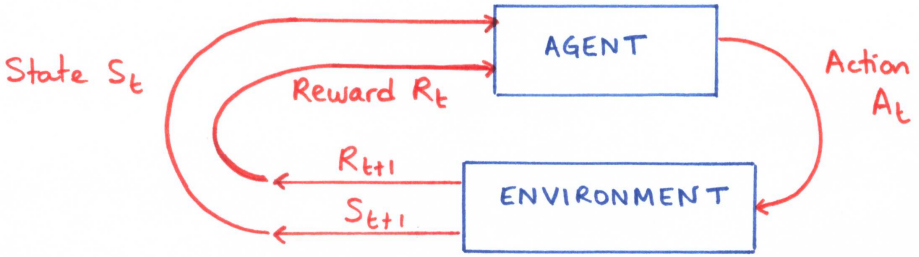
Markov Decision Processes (MDP) are ideal mathematical models for the RL problem.

→ The learner is called agent

→ Everything outside the agent is called the environment.

The agent & the environment continuously interact: at each step, the agent executes an action A_t, given some information about

the environment's state S_t. As a consequence of its action, the agent receives a reward R_{t+1}, and finds itself in a new state S_{t+1}



⇒ gives rise to a trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \dots$

We assume a finite MDP = states, actions, and rewards can only take finitely many values:

$S_t \in \mathcal{S}$
$A_t \in \mathcal{A}$
$R_t \in \mathcal{R}$

sets $\mathcal{S}, \mathcal{A}, \mathcal{R}$ contain finitely many elements.

→ Dynamics of the MDP: Random Variables R_t, S_t have discrete probability distributions that depend only on the preceding state & action (the Markov property):

$$p(s', r \mid s, a) := \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$$

↑ $\forall s, s' \in \mathcal{S}, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}$.

↑ sum to one.

Remark: the state representation is highly important to determine what happens next. If the state representation is not complete enough, then the Markov property is no longer justified, and the techniques developed may fail.

Ex: trajectory of a helicopter. The state variable should incorporate position, angular position, angular velocity, wind

direction, and so on. A state representation including position only is not sufficient to control the helicopter and chose actions that will lead it to some target: we would need to go back in time and consider its full trajectory (→ Markov property violated).

From the function p, we can compute important quantities such as

* state-transition probabilities.

$$p(s' \mid s, a) := \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) = \sum_{r \in \mathcal{R}} p(s', r \mid s, a).$$

* expected rewards for (state, action) pairs

$$r(s, a) := \mathbb{E}(R_{t+1} \mid S_t = s, A_t = a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a).$$

→ The sequence of rewards received after time t is $R_{t+1}, R_{t+2}, \dots$

We seek to maximize the (expected) return $\mathbb{E} G_t$, where

$$G_t = \underbrace{R_{t+1} + R_{t+2} + \dots + R_T}_{\text{cumulative reward}}$$

↑ finite horizon, finite time step.

In many cases, it is convenient to consider infinite horizon: $T = +\infty$. [Ex: robot with long life span]. The return formulation is problematic, since G_t could easily be infinite.

⇒ We need the concept of discounting.

The discounted return is: $G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$

$$= \sum_{k \geq 0} \gamma^k R_{t+k+1}, \quad \gamma \in [0, 1]$$

γ = discount rate

Why introducing γ ?

(i) Mathematically convenient : if $\gamma < 1$, the infinite sum converges provided that the reward sequence is bounded.

(ii) Trade-off between immediate reward (extreme case : $\gamma = 0$: "myopic" strategy as the agent is concerned only by its immediate rewards) and "far-sighted" objectives (as γ gets close to 1, the return G_t takes future rewards into account more strongly).

Note that $G_t = R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \dots)$

$$\Rightarrow \quad G_t = R_{t+1} + \gamma G_{t+1} \qquad (*)$$

↑ This relation will have strong implications later.

To summary, a MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$

$\mathcal{T}$: transitions $p(s' \mid s, a)$; γ : discount rate ; r : expected rewards $r(s, a)$

- A policy is a mapping from $\mathcal{S}$ to $\mathcal{A}$: $\pi : \mathcal{S} \to \mathcal{A}$
  - deterministic policy $a = \pi(s)$
  - stochastic policy $\pi(a \mid s) = \mathbb{P}(A = a \mid S = a)$ (stationary policy : does not depend on t)

↑ The agent's behaviour function.

- The value function v_π of a state s given a policy π quantifies the expected return when starting in s & following π.

$$v_\pi(s) := \mathbb{E}_\pi(G_t \mid S_t = s)$$

↑ ↑ Emphasizes the dependence of the value function on a particular π

- Another important quantity is the action-value function q_π, which characterizes the expected return when starting in s, taking action a, an following policy π :

$$q_\pi(s, a) := \mathbb{E}_\pi(G_t \mid S_t = s, A_t = a)$$

- A fundamental property of the value function & action-value function is that they can be decomposed into two parts :
  - immediate reward
  + discounted value of future reward :

From the decomposition $(*)$, we have

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi(R_{t+1} + \gamma G_{t+1} \mid S_t = s) \\ &= \mathbb{E}_\pi(R_{t+1} \mid S_t = s) + \gamma \mathbb{E}_\pi(G_{t+1} \mid S_t = s) \end{aligned}$$

$\mathbb{E}_\pi(R_{t+1} \mid S_t = s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) r(s, a) = \mathbb{E}_\pi\{r(s, A_t) \mid S_t = s\}$

Since $\mathbb{E}_\pi(R_{t+1} \mid S_t) = \mathbb{E}\{\mathbb{E}(R_{t+1} \mid S_t, A_t) \mid S_t\} = \mathbb{E}\{r(S_t, A_t) \mid S_t\}$

$\mathbb{E}_\pi(G_{t+1} \mid S_t) = \mathbb{E}_\pi \mathbb{E}_\pi(G_{t+1} \mid S_{t+1}, S_t) \mid S_t) = \mathbb{E}_\pi\{v_\pi(S_{t+1}) \mid S_t\}$

$$\mathbb{E}\{\underbrace{\mathbb{E}(v_\pi(S_{t+1}) \mid S_t = s, A_t = a)}_{= \text{function of } s, a = \sum_{s' \in \mathcal{S}} v_\pi(s') p(s' \mid s, a)} \mid S_t = s\}$$

$$= \text{function of } s = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} v_\pi(s') p(s' \mid s, a)$$

We obtain the decomposition:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, V_\pi(s') \right\}$$

aka the BELLMAN EQUATION for V_π.

With a deterministic policy, Bellman equation reduces to

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

$$V_\pi := \begin{pmatrix} | \\ V_\pi(s) \\ | \end{pmatrix} \qquad R_\pi := \begin{pmatrix} | \\ r(s, \pi(s)) \\ | \end{pmatrix}$$

In matrix notation:

$$\boxed{V_\pi = R_\pi + \gamma P_\pi V_\pi}$$

$$P_\pi := \begin{pmatrix} | \\ \text{---}\ p(s'|s,\pi(s))\ \text{---} \\ | \end{pmatrix} = |\mathcal{S}| \times |\mathcal{S}| \text{ transition matrix}$$

s-th raw $= (P_{s,s'})$

s'-th column

The (finite) system of linear equations $V_\pi = R_\pi + \gamma P_\pi V_\pi$ admits a unique solution $V_\pi^* = (I - \gamma P_\pi)^{-1} R_\pi$. To prove this, we only have to verify that $I - \gamma P_\pi$ is invertible.

$$\begin{aligned} \| P_\pi \|_\infty &= \max_s \sum_{s'} |P_{s,s'}| \\ &= \max_s \sum_{s'} \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = \pi(s)) \\ &= 1 \end{aligned}$$

$\Rightarrow \| \gamma P_\pi \|_\infty = \gamma < 1 \Rightarrow$ The eigenvalues of γP_π are all less than 1, and $(I - \gamma P_\pi)$ is thus invertible.

- Similarly,

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi(G_t \mid S_t = s, A_t = a) \\ &= \mathbb{E}_\pi(R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a) \\ &= \mathbb{E}_\pi(R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a) \\ &= r(s,a) + \gamma \underbrace{\mathbb{E}_\pi\{q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\}}_{\| \\ \sum_{s',a'} q(s',a')\, \mathbb{P}(S_{t+1} = s', A_{t+1} = a' \mid \substack{S_t = s \\ A_t = a}) \\ \| \\ \sum_{s',a'} q(s',a')\, \mathbb{P}(A_{t+1} = a' \mid \cancel{S_t = s}, \cancel{A_t = a}, S_{t+1} = s') \\ \qquad \Rightarrow \pi(a'|s') \\ \qquad \times \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \\ \qquad \Rightarrow p(s'|s,a) \\ \| \\ \sum_{s',a'} q(s',a')\, \pi(a'|s')\, p(s'|s,a) \\ \| \\ \sum_{s'} p(s'|s,a) \underbrace{\sum_{a'} \pi(a'|s')\, q(s',a')}_{\| \\ V_\pi(s')} \end{aligned}$$

Since

$$\begin{aligned} V_\pi(S_t) &= \mathbb{E}_\pi\{G_t \mid S_t\} \\ &= \mathbb{E}_\pi\{\mathbb{E}_\pi(G_t \mid S_t, A_t) \mid S_t\} \\ &= \mathbb{E}_\pi\{q_\pi(S_t, A_t) \mid S_t\} \end{aligned}$$

$$\Rightarrow V_\pi(s') = \sum_{a' \in \mathcal{A}} \pi(a'|s')\, q_\pi(s',a').$$

We finally obtain

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V_\pi(s')$$

BELLMAN EQUATION for q_π.

• Optimal policies & Optimal Value Functions.

→ A policy π is said to perform better than π' if $v_\pi(s) \geq v_{\pi'}(s) \quad \forall s \in \mathcal{S}$; we write $\pi \geq \pi'$.

There is always a policy that is better than or equal to all other policies ; but it may not be unique. If there is more than one, we denote by π_* all optimal policies. They all share the same value function v_* : $v_*(s) := \max_\pi v_\pi(s)$.

We see from Bellman equation for q_π that all optimal policies π_* also share the same optimal action-value function

$$\begin{aligned} q_*(s,a) &= r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s') \\ &= r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_\pi v_\pi(s) \\ &= \max_\pi \{ r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s) \} \\ &= \max_\pi q_\pi(s,a) \end{aligned}$$

Next, note that $v_*(s) = \sum_{a \in \mathcal{A}} \pi_*(a|s) \{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, v_*(s') \}$ ← Bellman equation for v_*

$$\begin{aligned} &= \sum_{a \in \mathcal{A}} \pi_*(a|s)\, q_*(s,a) \\ &\leq \sum_{a \in \mathcal{A}} \pi_*(a|s) \left(\max_a q_*(s,a) \right) \\ &= \max_a q_*(s,a) . \end{aligned}$$

In addition, if there exists a state s for which this inequality is strict : $v_*(s) < \max_a q_*(s,a)$, then taking a deterministic policy with the maximizing action would define a better policy ⇒ we must have equality : $v_*(s) = \max_a q_*(s,a) \quad \forall s$:

$$v_*(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s') \right\}$$

↖ BELLMAN EQUALITY EQUATION for v_*.

In addition, we observe that ← If we know q_*, we immediately know π_*. (and v_*)

$$\forall s \in \mathcal{S}, \quad \pi_*(s) = \operatorname*{argmax}_a q_*(s,a)$$

× Remark: Bellman optimality equation for q_* is

$$q_*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} q_*(s',a')$$

→ Bellman optimality equations are non-linear due to the presence of the max → in general, no closed form solution. However, many iterative solutions exist, and are presented in the next section.

× Remark: The knowledge of q_* make the choice of an optimal policy very easy. However, knowing v_* also allow us to deduce an optimal policy : for each s, there will be one or more actions a that achieve the maximum on the RHS of Bellman optimality equations for v_* ⇒ any policy that assigns a non-zero probability to these actions only is an optimal policy.

↑ Since for these values of a, Bellman equation is indeed satisfied :

$$v_*(s) = \underbrace{\sum_{a \in \mathcal{A}} \pi_*(a|s)}_{\text{sum to 1}} \underbrace{\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, v_*(s') \}}_{\max_a \{\dots\} = v_*(s)}$$

sum to 1 since π_* puts non-zero mass only on the values of a reaching the max.

II - DYNAMIC PROGRAMMING.

Dynamic Programming (DP) refers to a family of algorithms that can be used to compute optimal policies given the MDP parameters. (term DP is due to Bellman '57)

↳ No learning: $p(s', r \mid s, a)$ is given.

⇒ A PLANNING problem.

We discuss in this section two planning algorithms: policy iteration & value iteration.

II.1. Policy Evaluation.

We first address a simple problem: given a policy π (deterministic or stochastic), how to compute the state-value function $v_\pi(s)$, $\forall s \in \mathcal{S}$?

→ Recall Bellman equation from page 7:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) v_\pi(s') \right\}$$

(*)

$$V_\pi = R_\pi + \gamma P_\pi V_\pi \quad \text{(matrix form)}$$

- V_π ($|\mathcal{S}| \times 1$) $= \begin{pmatrix} | \\ v_\pi(s) \\ | \end{pmatrix} \leftarrow s$-th row
- P_π ($|\mathcal{S}| \times |\mathcal{S}|$) $= \begin{pmatrix} | \\ \cdots \sum_{a \in \mathcal{A}} \pi(a \mid s) p(s' \mid s, a) \cdots \\ | \end{pmatrix} \leftarrow s$-th row, with s'-th column
- R_π ($|\mathcal{S}| \times 1$) $= \begin{pmatrix} | \\ \sum_{a \in \mathcal{A}} \pi(a \mid s) r(s,a) \\ | \end{pmatrix} \leftarrow s$-th row

(*) = system of $|\mathcal{S}|$ linear equations with $|\mathcal{S}|$ unknowns.

It can be solved

- ↘ using matrix inversion: $V_\pi = (I - \gamma P_\pi)^{-1} R_\pi$
- ↘ iteratively: $V_\pi^{(k+1)} = R_\pi + \gamma P_\pi V_\pi^{(k)}$

↑

Note that $\| P_\pi \|_\infty = \max_s \sum_{s'} |p_{s,s'}|$

$$= \max_s \sum_{s'} \sum_a \pi(a \mid s) p(s' \mid s, a)$$

$$= \max_s \underbrace{\sum_a \pi(a \mid s) \underbrace{\sum_{s'} p(s' \mid s, a)}_{=1}}_{=1} = 1$$

$\Rightarrow \| \gamma P_\pi \|_\infty = \gamma < 1 \Rightarrow$ Eigenvalues of γP_π are all < 1.

This ensures that (i) the matrix $(I - \gamma P_\pi)$ is invertible

(ii) sequence $V_\pi^{(k)}$ converges to the solution.

- Direct inversion of $(I - \gamma P_\pi)$ may be tedious & computationally expensive, and we use the iteration method, more suitable for our purposes:

ITERATIVE POLICY EVALUATION

$\forall s \in \mathcal{S}$, compute (until $\max_s |v^{(k+1)}(s) - v^{(k)}(s)| < \varepsilon$)

$$v_\pi^{(k+1)}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) v_\pi^{(k)}(s') \right\}$$

↑ All states are updated using the old value $v_\pi^{(k)}$. aka SYNCHRONOUS UPDATES. Alternatively, after one state s is updated, the new value function may be plugged back into the right-hand side, overwriting the old one → asynchronous updates.

Ex: Consider the following 4 × 4 grid :

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

terminal states : if you arrive here, you stay here

- States : Where on the grid you are.
- Actions : Move North, South, East or West only
  (from 6, you can only get to 2, 5, 7, 10)
  (from 7, you can get to 3, 6, 11, or remain at 7, shall you decide to move East).
  (from 3, you can go to 2 or 7, or remain at 3 shall you decide to go North or East).
- Rewards : $R_t = -1$ on all transitions.
- Policy : Uniform policy : moves N, S, E or W occur with probability 1/4 : $\pi(a \mid s) = 1/4$ for $a \in \{N, S, E, W\}$.

↳ Transition probabilities are of the form :

$p(5, -1 \mid 9, \text{north}) = 1$

(s', r, s, a) (notation from page 3)

$p(6, r \mid 13, \text{south}) = 0 \quad \forall r$ .../...

↳ Goal : Evaluate $v_\pi(s)$ for all s (i.e. for all positions on the grid)

↳ Use Iterative Policy Evaluation Algorithm.

Put R_π (16×1) $:= \begin{pmatrix} 0 \\ -1 \\ \vdots \\ -1 \\ 0 \end{pmatrix}$ ← terminal state ▨ = no reward ; ↕ 14 entries (fixed rewards = −1) ; ← terminal state ▨ = no reward

P_π = (from state s ↓ / to state s' →)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1/4	1/4	1/4		1/4									
2		1/4	1/4	1/4		1/4								
3			1/4	1/2			1/4							
4	1/4				1/4	1/4		1/4						
5		1/4			1/4		1/4		1/4					
6			1/4		1/4		1/4			1/4				
7				1/4		1/4	1/4				1/4			
8					1/4			1/4	1/4			1/4		
9					1/4			1/4		1/4			1/4	
10						1/4			1/4		1/4			1/4
11							1/4			1/4	1/4			1/4
12								1/4				1/2	1/4	
13									1/4			1/4	1/4	1/4
14										1/4			1/4	1/4	1/4

= 0 everywhere else

Each entry is equal to $\sum_{a \in A} \pi(a \mid s)\, p(s' \mid s, a)$ ($\pi(a \mid s) = 1/4$; $p(s' \mid s, a)$: all 0/1 proba)

Take $V_\pi^{(0)}$ (16×1) $= \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$ & put $V_\pi^{(k+1)} = R_\pi + \gamma P_\pi V_\pi^{(k)}$ $k = 0, 1, \ldots$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$k=1$

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

$k=2$

0	-2.44	-2.94	-3
-2.44	-2.87	-3	-2.94
-2.94	-3	-2.87	-2.44
-3	-2.94	-2.44	0

$k=3$

Value of the uniform policy after k iterations ↗

0	-3.66	-4.69	-4.91
-3.66	-4.48	-4.78	-4.69
-4.69	-4.78	-4.48	-3.66
-4.91	-4.69	-3.66	0

$k=5$

0	-6.14	-8.35	-8.97
-6.14	-7.74	-8.43	-8.35
-8.35	-8.43	-7.74	-6.14
-8.97	-8.35	-6.14	0

$k=10$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

$k=\infty$ (solution $V_\pi = (I-\gamma P_\pi)^{-1} R_\pi$)

Interpretation: each number in each cell represents the average number of steps starting from that cell, before reaching a terminal absorbing state.

II.2. Policy Improvement.

Suppose we have determined the value v_π of a policy π. How to change its value at some state s to find a better policy

- Tool: the action-value function. Recall:

$$q_\pi(s,a) = \underbrace{r(s,a) + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')}$$

Interpretation: value of taking action a at state s, and the following policy π.
→ If you find a value a such that $q_\pi(s,a)$ is larger than $v_\pi(s)$, i.e. if it is better to select action a once from state s & then follow policy π, then you would expect that you are better off following action a from s all the time. And indeed, we have the following result:

Theorem. Consider a deterministic policy π, π'

If $\forall s \in \mathcal{S}$ $\quad q_\pi(s, \pi'(s)) \geq v_\pi(s)$. ——— (1)

Then

$\forall s \in \mathcal{S}$ $\quad v_{\pi'}(s) \geq v_\pi(s)$ ——— (2)

In addition, if there is at least one strict inequality at (1), the inequality at (2) is also strict.

proof:

$$v_\pi(s) \leq q_\pi(s, \pi'(s))$$
$$= r(s, \pi'(s)) + \gamma \sum_{s'} p(s'|s, \pi'(s)) \underbrace{V_\pi(s')}_{\leq q_\pi(s', \pi'(s'))}$$
$$\leq r(s, \pi'(s)) + \gamma \sum_{s'} p(s'|s, \pi'(s)) \underbrace{q_\pi(s', \pi'(s'))}_{=}$$
$$r(s', \pi'(s')) + \gamma \sum_{s''} p(s''|s', \pi'(s')) V_\pi(s'')$$
$$= r(s, \pi'(s)) + \gamma \sum_{s'} p(s'|s, \pi'(s)) r(s', \pi'(s'))$$
$$+ \gamma^2 \sum_{s', s''} p(s'|s, \pi'(s)) p(s''|s', \pi'(s')) \underbrace{V_\pi(s'')}_{\leq q_\pi(s'', \pi'(s''))}$$

$\mathbb{E}\{R_{t+1} | S_t = s\}$

$\mathbb{E}\{R_{t+2} | S_t = s\}$

... /... keep iterating

$$\leq v_{\pi'}(s) \quad \left(= \mathbb{E}\left(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \dots \,|\, S_t = s\right)\right)$$

→ Instead of changing a single state s only, it is natural to consider changes at all states, and consider the new GREEDY policy π':

POLICY IMPROVEMENT

$$\forall s \in \mathcal{S}$$
$$\pi'(s) = \underset{a}{\operatorname{argmax}}\ q_\pi(s,a)$$
$$= \underset{a}{\operatorname{argmax}} \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s') \right\}$$

→ Suppose now that the new policy π' is as good as π, but not strictly better: $v_{\pi'}(s) = v_\pi(s)\ \forall s$. Then we have

$$v_{\pi'}(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_{\pi'}(s') \right\}$$
$$\uparrow \qquad\qquad (v''_\pi(s'))$$

This is exactly Bellman inequality equation.

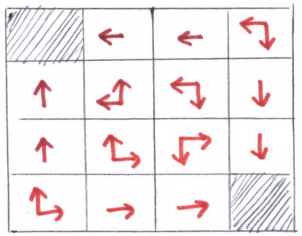
⇒ $v_{\pi'} = v_\pi = v^*$ & both π / π' are optimal policies.

⇒ Policy improvement is guaranteed to give us a strictly better policy, unless the optimal policy is reached.

Ex: Back to the grid example on page 13. The uniform policy has value

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

greedy policy improvement



policy π'.

In this example, the improved policy π' turns out to be optimal. This is not true in general.

II.3. Policy Iteration.

The policy iteration algorithm alternates between policy evaluation & greedy policy improvement, generating a sequence

$$\pi_0 \to v_{\pi_0} \to \pi_1 \to v_{\pi_1} \to \pi_2 \to v_{\pi_2} \to \dots$$

Each iteration must be a strict improvement of the policy, unless we are at the optimal. Since a finite MDP has finitely many policies, the sequence must converge to the optimal policy in a finite number of steps.

- Goal: compute the optimal policy π^*.
- Init: arbitrary policy π_0
  $\pi \leftarrow \pi_0$
  $\pi' \leftarrow$ NIL
- While $(\pi \neq \pi')$ do
  - ↘ Policy Evaluation: determine v_π (page 12); $V \leftarrow v_\pi$
  - ↘ $\pi' = \pi$
  - ↘ Greedy Policy Improvement:
    $\pi = \underset{\pi}{\operatorname{argmax}} \{ R_\pi + \gamma P_\pi V \}$
- Return π

POLICY ITERATION

II. 4. Value Iteration.

- Given a policy π with value function $v_\pi(s)$ (= $r(s,\pi(s)) + \gamma \sum_{s'} p(s'|s,\pi(s))\, v_\pi(s')$), we obtain a better policy π' by selecting for each s, the value a maximizing $r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s')$.

  ↳ Back to the grid example, the value function of the random policy ¼ ← ¼ ↑ (π) ↓ ¼ can be used to greedily select a better policy π' (which is optimal in this particular example). However, if the value function v_π is calculated iteratively, we see that a greedy policy based on the third iterate ($k=3$) yields already the same optimal policy π' (see page 14). This observation shows that we may not need to wait until convergence of the iterative policy evaluation algorithm to greedily select the next policy; which raises the question: after how many iterations stop?

- The value iteration algorithm stops after exactly one iteration during the evaluation step:

  → Given $v^{(0)}(s)$ [initialization: arbitrary value]

  → Calculate (policy improvement algo with early stopping)

$$v^{(1)}(s) = \max_\pi \Big\{ r(s,\pi(s)) + \gamma \sum_{s'} p(s'|s,\pi(s))\, v^{(0)}(s') \Big\}$$

  1. Evaluation step (stop after 1st iteration)

  2. Improvement step: greedy search.

  deterministic policy

& repeat:

$$v^{(k+1)}(s) = \max_\pi \Big\{ r(s,\pi(s)) + \gamma \sum_{s'} p(s'|s,\pi(s))\, v^{(k)}(s') \Big\}$$

↑ Another way to understand this expression is by reference to Bellman optimality equation for v_* (see page 9): the Bellman equation is simply made recursive.

↘ In matrix notation: $V^{(k+1)} = \max_\pi \{ R_\pi + \gamma P_\pi V^{(k)} \} =: \Phi(V^{(k)})$ ($|\mathcal{S}| \times 1$)

Theorem: For any initial value $V^{(0)}$, the sequence of iterates $V^{(k+1)} = \Phi(V^{(k)})$ converges to the optimal value function V^*.

proof: We show that Φ is γ-Lipschitz for $\|\cdot\|_\infty$. Since $\gamma < 1$, Φ is then a contraction & admits a fixed point; and any sequence of iterates converges to the fixed point.

Let $s \in \mathcal{S}$, $V \in \mathbb{R}^{|\mathcal{S}|}$, $U \in \mathbb{R}^{|\mathcal{S}|}$,

& denote by $a^*(s) = \operatorname{argmax}_a \{ r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s') \}$

Then

$$\Phi(V)(s) - \Phi(U)(s) \leq \Phi(V)(s) - \Big\{ r(s,a^*(s)) + \gamma \sum_{s'} p(s'|s,a^*(s))\, U(s') \Big\}$$

$$= \gamma \sum_{s'} p(s'|s,a^*(s)) \{ V(s') - U(s') \}$$

$$\leq \gamma \sum_{s'} p(s'|s,a^*(s)) \, \|V - U\|_\infty$$

$$= \gamma \, \|V - U\|_\infty .$$

$\Phi(V)(s)$: s-th entry of the vector $\Phi(V)$

& similarly $\Phi(U)(s) - \Phi(V)(s) \leq \gamma \| V - U \|_\infty$,

so that $|\Phi(V)(s) - \Phi(U)(s)| \leq \gamma \| V - U \|_\infty$.

↑ Φ is γ-Lipschitz.

Thus

$\| V^* - V^{(k+1)} \|_\infty = \| \Phi(V^*) - \Phi(V^{(k)}) \|_\infty$

$V^* = \Phi(V^*)$

$\leq \gamma \| V^* - V^{(k)} \|_\infty$

$\leq \gamma^{k+1} \| V^* - V^{(0)} \|_\infty$.

$\to 0$ as $k \to +\infty$. ∎

Remarks. (i) Intermediate value functions may not correspond to any policy: it can happen that $\nexists \pi$ such that $v^{(k)} = v_\pi$.

(ii) The algorithm $V^{(k+1)} = \Phi(V^{(k)})$ uses synchronous backups. Instead, we may use asynchronous backups.

(iii) Policy & Value Iteration algorithms are based on state-value functions. The complexity is $O(m|\mathcal{S}|^2)$ per iteration, for m actions and $|\mathcal{S}|$ states. We could apply the same ideas to the action-value functions. Complexity is $O(m^2|\mathcal{S}|^2)$.

For each state, we need to consider the successor state ($\to |\mathcal{S}|^2$ operations), and there are m actions.

- Goal: compute the optimal policy π^* / value function v^*
- Init: $V^{(0)}$
- Repeat $k = 0, 1, \dots$

$$V^{(k+1)} = \max_\pi \{ R_\pi + \gamma P_\pi V^{(k)} \}$$

- Until Convergence.

VALUE ITERATION

III - MODEL-FREE PREDICTION

MDP environment but we do not know the model parameters (i.e. transition & reward probabilities).

↳ We want to find the optimal way to behave.

In this section, we give methods to estimate the value function of a given policy, when the environment is unknown.

In section IV, we optimize the value function of an unknown MDP (control).

We discuss two classes of techniques:
- MC learning
- TD learning.

These methods all learn from experience: given a policy π, sample trajectories according to π, observe rewards & successive states sampled by the environment, and update the value function estimate based on sample returns.

↑ learning from actual experience

III.1. Monte Carlo Methods.

- Assumption: Experience is divided into episodes, and all episodes eventually terminate, no matter what actions are performed.

Episode #1 : $S_0^{(1)}, A_0^{(1)}, R_1^{(1)}, S_1^{(1)}, A_1^{(1)}, R_2^{(1)}$ - end

Episode #2 : $S_0^{(2)}, A_0^{(2)}, R_1^{(2)}$ - end

Episode #3 : $S_0^{(3)}, A_0^{(3)}, R_1^{(3)}, S_1^{(3)}, A_1^{(3)}, R_2^{(3)}$ - end

⋮

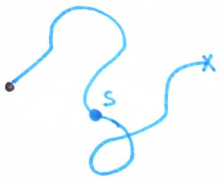
Actions are selected by the agent; states & rewards are received by the environment (no need to know its dynamics)

At the end on an episode, the value estimates are updated (episode-to-episode updates; not "on-line").

Remark: Learning is from complete episodes only.

- Suppose we wish to estimate the value $v_\pi(s)$ of policy π at state s.
  Let t = the first time in an episode that state s is visited (s can be visited multiple times).

  Ex: $s' \to a' \to r' \to (s) \to a'' \to r' \to s' \to a' \to r'$
  $\to (s) \to a'' \to r''' -$ end
  ↖ first visit to s
  ↖ second visit to s.



↘ Return at time t is $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$.

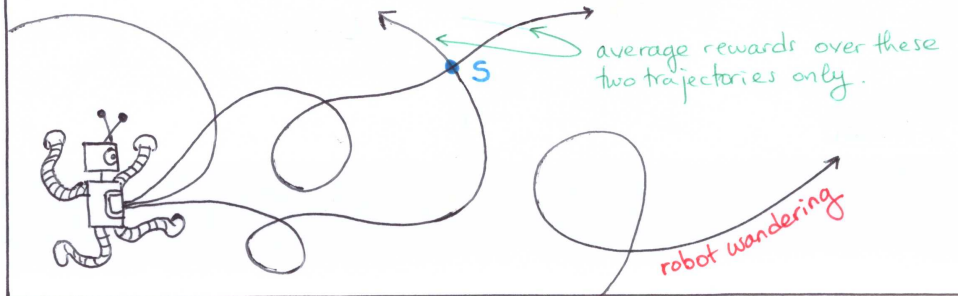
& the value function at state s is

$v_\pi(s) = \mathbb{E}(G_t \mid S_t = s)$

↖ Monte-Carlo (MC) techniques use the empirical mean returns instead of the population mean. (SLLN ensure convergence)

- 2 paradigms:

  ↘ "First-visit" MC policy prediction considers the reward G_t associated with the first-time step t that state s is visited. The value $v_\pi(s)$ is estimated by sample mean return.



Algorithm: First-visit MC prediction

- Goal: Estimate v_π of a policy π (unknown environment)
- Init: . $V(s) \in \mathbb{R}$ arbitrary $\forall s \in \mathcal{S}$.
  . Returns ← empty list $\forall s \in \mathcal{S}$.
- Repeat (for each episode)
  (1) Generate episode following policy π:
  $S_0, A_0, R_1, S_1, A_1, R_2, \cdots, S_{T-1}, A_{T-1}, R_T$.
  (2) $G \leftarrow 0$
  (3) For $t = T-1, T-2, \ldots, 0$:
  $G \leftarrow \gamma G + R_{t+1}$
  Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
  Add G to Returns(S_t)
  Update $V(S_t) \leftarrow$ Average(Returns(S_t)).

↘ "Every-visit" MC policy prediction averages the returns following every visit to s. The "every-visit" algorithm is the same as the algorithm presented above, but without the line "Unless S_t appears in $S_0, \ldots, S_{t-1}$".

- Result: Both first-visit & every-visit MC converge to $v_\pi(s)$ as the number of visits to s goes to infinity.
  Fist-visit → A direct consequence of the SLLN (a.s. convergence)
  Every-visit → see Singh & Sutton (1996)

- Caveat: We have to wait until the end of an episode to compute the mean. We present next TD learning, which learns from incomplete episodes, by bootstrapping.

III.2. TD learning.

- Temporal-Difference (TD) learning is a combination of MC ideas & Dynamic Programming (DP) ideas.
  - ↗ MC : learn from experience without knowing the environment's dynamics.
  - ↘ DP : update estimates from already learned estimates, without waiting for the final outcome.

In MC learning, the value of a state s is estimated by averaging the returns G_t :

"Update $v(S_t) \leftarrow$ Average(Returns(S_t))."

Instead of recomputing the sample mean at each iteration, the mean can be calculated incrementally:

- Let $X_0, X_1, \dots, X_n$ iid. Then

$$\bar{X}_{n+1} := \frac{1}{n+1}\sum_{i=1}^{n+1} X_i = \frac{1}{n+1}\left\{\sum_{i=1}^{n} X_i + X_{n+1}\right\}$$

$$= \frac{n}{n+1}\left\{\frac{1}{n}\sum_{i=1}^{n} X_i\right\} + \frac{1}{n+1} X_{n+1}$$

$$(*) \quad \boxed{\bar{X}_{n+1} = \left(1 - \frac{1}{n+1}\right)\bar{X}_n + \frac{1}{n+1} X_{n+1}}$$

$\xrightarrow{\text{a.s.}} \mathbb{E}X$ as $n \to +\infty$.

More generally, a.s. convergence of $\bar{X}_n$ to $\mathbb{E}X$ holds for

$$(**) \quad \boxed{\bar{X}_{n+1} = (1 - \alpha_n)\bar{X}_n + \alpha_n X_{n+1}},$$

provided the sequence $\{\alpha_n\}$ of positive numbers satisfies $\sum_{j\geq 0} \alpha_j = +\infty$; and $\sum_{j\geq 0} \alpha_j^2 < +\infty$.

(see e.g. thm 14.5 in Mohri & al, Foundations of Machine Learning).

We may rewrite (**) slightly differently:

$$\bar{X}_{n+1} = \bar{X}_n + \alpha_n(X_{n+1} - \bar{X}_n)$$

$$= \bar{X}_n + \alpha_n(\underbrace{X_{n+1} - \mathbb{E}X}_{\| \; Z_{n+1}} + \underbrace{\mathbb{E}X - \bar{X}_n}_{\| \; f(\bar{X}_n)})$$

(zero mean)(iid) with $f(x) = \mathbb{E}X - x$

$$= \bar{X}_n + \alpha_n(f(\bar{X}_n) + Z_{n+1})$$

$$(***) \quad \boxed{\bar{X}_{n+1} = \bar{X}_n + \alpha_n F(\bar{X}_n, Z_{n+1})} \qquad F(x,z) := f(x) + z$$

$\xrightarrow{\text{a.s.}}$ to the root of $f(x) = \mathbb{E}_z\{F(x,z)\}$

↑ since $\mathbb{E}_z Z = 0$ & $f(x) = \mathbb{E}X - x$ & $F(x,z) := f(x) + z$

Convergence of the sequence $\{\bar{X}_n\}$ to the root of $f(x) = \mathbb{E}_z\{F(x,z)\}$ holds under general assumptions on F, and on the sequence α_n.

↳ aka Stochastic Approximation Algorithms

↳ dates back to Robbins & Monro (1951).

Ex: We prove in the appendix of the Chapter SL = GRADIENT DESCENT ALGORITHMS that assuming

(a) $F : \mathbb{R}^2 \to \mathbb{R}$ s.t. $\forall x \in \mathbb{R}$ $\forall z \in \mathbb{R}$ $|F(x,z)| \leq K$ (boundedness)

(b) $Z_1, \dots, Z_n, \dots$ iid with distribution G.

(c) $f(x) = \mathbb{E}_z\{F(x,z)\}$ is continuous, strictly ↘, with a unique $x^* \in \mathbb{R}$ such that $f(x^*) = 0$.

(d) $\alpha_n > 0$, $\sum_{j\geq 0} \alpha_j = +\infty$, $\sum_{j\geq 0} \alpha_j^2 < +\infty$

(e) X_0 is square integrable.

Then the sequence $X_{n+1} = X_n + \alpha_n F(X_n, Z_{n+1})$ converges a.s. to x^*.

Assumptions (a) – (e) can be weakened considerably. For example, instead of boundedness, we may assume that $F: \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$ is such that $\forall x \in \mathbb{R}^d \quad \mathbb{E}|F(x,Z)| \leq K(1+\|x\|^2)$, for some $K > 0$

- Application of these ideas to MDP.

Recall that the state-value function v_π satisfies Bellman equation:

$$v_\pi(s) = r(s,\pi(s)) + \gamma \sum_{s'} p(s'|s,\pi(s))\, v_\pi(s')$$

$$= \mathbb{E}_{S_{t+1}}\{ r(s,\pi(s)) + \gamma\, v_\pi(S_{t+1}) \mid S_t = s \}$$

$$\Updownarrow$$

$$\mathbb{E}_{S_{t+1}}\{ \underbrace{r(s,\pi(s)) + \gamma\, v_\pi(S_{t+1}) - v_\pi(s)}_{F(V_\pi, S_{t+1})} \mid S_t = s \} = 0$$

↑ vector of dim $|\mathcal{S}|$ $\quad V_\pi = \begin{pmatrix} \vdots \\ v_\pi(s) \\ \vdots \end{pmatrix} \leftarrow s\text{-th row}$

& we are precisely looking for the root of the function $\mathbb{E}_{S_{t+1}}\{ F(V_\pi, S) \mid S = s \}$, $s \in \mathcal{S}$. (function of V_π)

⇒ Consider the sequence of estimates:

$$v_\pi(S_t) \leftarrow v_\pi(S_t) + \alpha\{ r(S_t, \pi(S_t)) + \gamma\, v_\pi(S_{t+1}) - v_\pi(S_t) \}$$

Updates are "on-line": no need to wait until the end of an episode.

temporal differences of value functions

→ This procedure is known as TD(0).

Algorithm: TD(0)

× Goal: Estimate v_π of a policy π (unknown environment)

× Init: $V_\pi(s) \in \mathbb{R}$ arbitrary except v_π(terminal states) $= 0$

× Repeat (for each episode)

(i) Initialize S

(ii) Repeat for each step of episode:

- $A \leftarrow$ Action given by π for S
- Take action A, observe R & S'
- $v_\pi(S) \leftarrow v_\pi(S) + \alpha(R + \gamma\, v_\pi(S') - v_\pi(S))$
- $S \leftarrow S'$

Until S is a terminal state.

"TD updates a guess towards a guess".

⇒ TD learns at each iteration, before knowing the final outcome. It learns from incomplete episodes by taking advantage of the Markov dynamics of the MDP model.

- Compare:

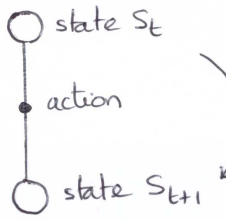
$$v_\pi(s) = \mathbb{E}(G_t \mid S_t = s)$$

final reward → MC (no bias, but high variance)

$$= \mathbb{E}(R_{t+1} + \gamma\, G_{t+1} \mid S_t = s)$$

$$= \mathbb{E}(R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s)$$

immediate reward + discounted value of the next step → TD (biased, but lower variance)



one-step bootstrapping

The procedure can be generalized, allowing n-step bootstrapping instead.

- n-step bootstrapping simply iterates the previous idea:

$G_t = R_{t+1} + \gamma G_{t+1}$

$= R_{t+1} + \gamma R_{t+2} + \gamma^2 G_{t+2}$, and so on,

.../---

The n-step return is

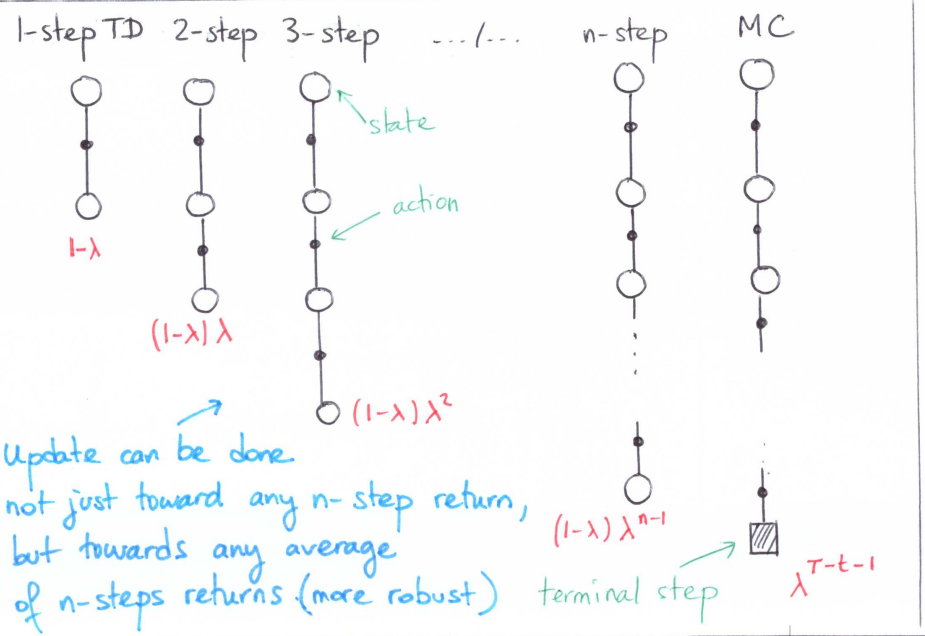
$G_t^n := R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v_\pi(S_{t+n})$

= full return truncated after n steps, & corrected for the remaining missing terms by $v_\pi(S_{t+n})$

↳ To compute G_t^n, we need future rewards $R_{t+1}, \ldots, R_{t+n}$. The first time these are available is $t+n \Rightarrow$ the n-step TD algorithm updates according to:

$v_\pi(S_t) \leftarrow v_\pi(S_t) + \alpha \{ G_t^n - v_\pi(S_t) \}$

(no updates are made during the first n-1 steps of each episode).



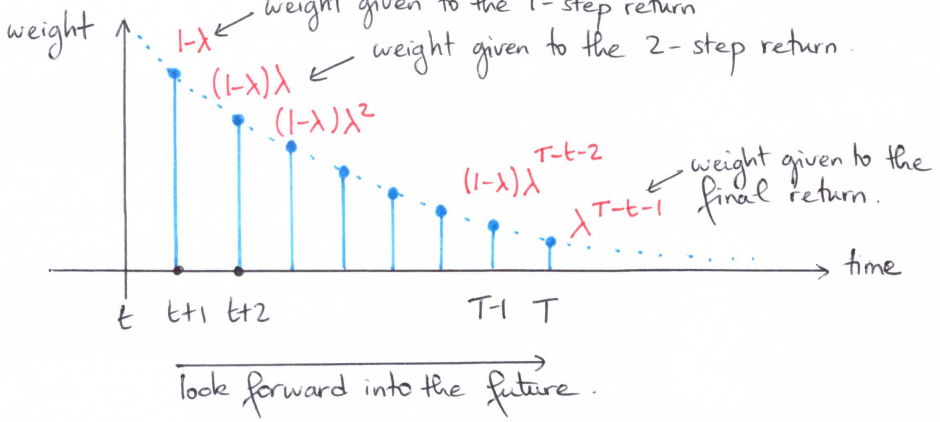
Update can be done not just toward any n-step return, but towards any average of n-steps returns (more robust)

Averaging the n-step returns can be done in many different ways. The λ-return G_t^λ combines all n-step returns G_t^n using weights $(1-\lambda)\lambda^{n-1}$:

$$G_t^\lambda := (1-\lambda) \sum_{n=1}^{+\infty} \lambda^{n-1} G_t^n .$$

In practice, episodes terminates, and the weight given to the final return is λ^{T-t-1}, to ensure that weights sum to 1.

- The λ-return corresponds to a geometrically weighted average.



- The TD(λ) algorithm updates the value-function estimate according to

$$v_\pi(s) \leftarrow v_\pi(s) + \alpha \{ G_t^\lambda - v_\pi(s) \}$$

TD(λ) update

Like MC, the TD(λ) update can be made only at the end of an episode.

- In practice, take α small ≈ 0.1 - 0.3

IV. MODEL-FREE CONTROL.

We consider policy improvement in an MDP environment, with unknown model parameters. We discuss two algorithms

(i) SARSA

(ii) Q-learning

IV.1. SARSA.

- Recall the Bellman equation satisfied by the action-value function q_π:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s') \qquad \text{(p. 8)}$$

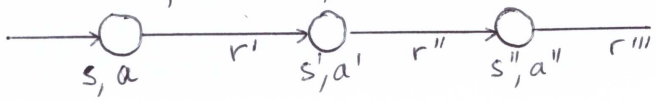
If we knew the model dynamics, we could improve on policy π by acting greedily on q_π; and select

$$\pi'(s) = \operatorname*{argmax}_a q_\pi(s,a) \qquad \text{(p. 12)}$$
$$= \operatorname*{argmax}_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s') \right\} \quad (*)$$

In particular, this ensures that $v_{\pi'}(s) \geq v_\pi(s)$. (p. 16)

Making this process iterative yields the policy iteration algorithm, of which the value iteration algorithm can be seen as a special case.

- In an unknown environment, the max in (*) cannot be calculated, and to improve (& evaluate) a policy π, one must learn from experience. Consider a deterministic policy π. Generate trajectories / episodes:



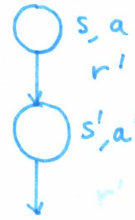
The sequence of observed values (states + rewards) associated with policy π is used to estimate q_π using a stochastic approximation algorithm. Bellman equation for q_π can be rewritten

$$\mathbb{E}_{S_{t+1}}\left\{ r(s,a) + \gamma \underbrace{v_\pi(S_{t+1})}_{q_\pi(S_{t+1}, \pi(S_{t+1}))} - q_\pi(s,a) \,\middle|\, \begin{matrix} S_t = s \\ A_t = a \end{matrix} \right\} = 0$$

$$\mathbb{E}_{S_{t+1}}\left\{ r(s,a) + \gamma\, q_\pi(S_{t+1}, \pi(S_{t+1})) - q_\pi(s,a) \,\middle|\, \begin{matrix} S_t = s \\ A_t = a \end{matrix} \right\} = 0$$

Consider

$$q(s,a) \leftarrow q(s,a) + \alpha\left\{ r' + \gamma q(s',a') - q(s,a) \right\} \qquad (*)$$



where $\begin{cases} a = \pi(s) \\ a' = \pi(s') \end{cases}$ ↖ deterministic here, but can be stochastic in general.

Keep iterating until convergence, and then act greedily: select π' such that $\pi'(s) = \operatorname*{argmax}_a q_\pi(s,a)$ ↖ your estimate

We can improve on this procedure in several directions.

(a) Speed up the procedure:

No need to wait until converge to act greedily on q_π.

Act greedily after each iteration / update of the action value function:

- Given π = current policy, select $a \sim \pi$, $a' \sim \pi$
- update $q(s,a)$ [one iteration of (*)]
- update policy $\pi \to \pi'$ by acting greedily on the new q

(b) Exploration vs Exploitation.

Some pairs (state, action) will not be seen/evaluated by acting greedily ⇒ we are unsure that we end up with the optimal policy ⇒ local minima.

The simplest idea for ensuring a continual exploration is to select the greedy action with probability $1-\varepsilon$.

$$\pi'(a|s) = \begin{cases} \varepsilon/|\mathcal{A}| + 1 - \varepsilon & \text{if } a = \operatorname{augmax}_{a'} q(s,a') \\ \varepsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$$

↑ The $|\mathcal{A}|$ actions are tried with non-zero probability.

↳ Terminology: we say that we act "ε-greedily".

Note that for any ε-greedy policy π, the ε-greedy policy π' with respect to q_π is an improvement; i.e. $v_{\pi'}(s) \geq v_\pi(s)$.

Indeed, first note that

$$q_\pi(s, \pi'(s)) = \sum_{a \in \mathcal{A}} \pi'(a|s)\, q_\pi(s,a)$$

$$= \frac{\varepsilon}{|\mathcal{A}|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a)$$

The max is ≥ than any weighted sum of its components, provided the weights are ≥0 & sum to 1 (indeed, weights are ≥0 since π is an ε-greedy policy & sum to 1 since $\sum_a \pi(a|s) = 1$)

$$\geq \frac{\varepsilon}{|\mathcal{A}|} \sum_a q_\pi(s,a) + (1-\varepsilon) \sum_a \left\{ \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}|}}{1-\varepsilon} \right\} \times q_\pi(s,a)$$

$$= \sum_a \pi(a|s)\, q_\pi(s,a)$$

$$= v_\pi(s).$$

+ use the result on page 16 to conclude that $v_{\pi'}(s) \geq v_\pi(s)$. ■

× Algorithm : SARSA.

× Goal : policy improvement in unknown environment

× Init : $q \leftarrow q_0$ for an arbitrary initial q_0.

× Repeat (until convergence) $t = 0, 1, \ldots$

$s \leftarrow$ Select state

$a \leftarrow$ Select action $\sim \pi$ from q, ε-greedily

For each epoch t, do

$r' \leftarrow$ reward (s, a)

$s' \leftarrow$ next state (s, a)

$a' \leftarrow$ select action $\sim \pi_t$ from q, ε_t-greedily

$q(s,a) \leftarrow q(s,a) + \alpha_t(r' + \gamma\, q(s',a') - q(s,a))$

$s \leftarrow s'$

$a \leftarrow a'$

× Return q

Remarks: (i) The name of the algorithm comes from the order of successive actions taken $s - a - r' - s' - a'$.

(ii) SARSA is an example of an on-policy algorithm; meaning that the update on q depends on the learning policy (action a' is selected from π). In contrast, the Q-learning algorithm presented in the next section is an example of an off-policy algorithm.

(iii) Convergence of SARSA.

SARSA converges to the optimal action-value function

provided

- $\sum \alpha_t = +\infty$; $\sum \alpha_t^2 < +\infty$ (convergence of the stochastic approximation algorithm → Robbins-Monro)
- Greedy in the Limit with Infinite Exploration (GLIE) sequence of policies $\pi_t(a|s)$

meaning : we must explore (all pairs (s,a) must be visited infinitely many times) ; but we must be greedy in the limit to satisfy Bellman & get to the optimal policy (≡ stop exploring)

Take e.g. $\varepsilon_t \sim \frac{1}{t}$.

(iii) Consider instead n-step returns $R_{t+1} + \gamma R_{t+2} + \dots + \gamma^n q(S_{t+n}, A_{t+n})$

↳ n-step SARSA .

+ consider weighted averages of all n-step retur

↳ SARSA (λ) algorithm

↑ Same idea as going from TD(0) to TD(λ).

IV. 2. Q-learning.

The alternative to SARSA is to start from Bellman optimality equation for q_* :

$$q_*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} \{ q_*(s',a') \}$$

$$\Updownarrow$$

$$\mathbb{E}_{S_{t+1}} \left\{ r(s, a) + \gamma \max_{a'} (q_*(S_{t+1}, a')) - q_*(s,a) \,\middle|\, \begin{matrix} S_t = s \\ A_t = a \end{matrix} \right\} = 0$$

↑ Solve this recursively using a stochastic approx. algorithm.

Consider

$$q(s,a) \leftarrow q(s,a) + \alpha \{ r' + \gamma \max_{a'} q(s,a') - q(s,a) \}$$

↑ An "off-policy" algorithm : max is taken over all actions a' : a' is not selected from the current policy (compare with SARSA) .

- Algorithm : Q-learning
- Goal : policy improvement in an unknown environment
- Init : $q \leftarrow q_0$ for an arbitrary initial q_0
- For $t = 0, 1, \dots$, do

  $s \leftarrow$ select state

  For each epoch t do

  $a \leftarrow$ select action $\sim \pi$ from q, ε_t-greedily

  $r' \leftarrow$ reward (s,a)

  $s' \leftarrow$ next state (s,a)

  $q(s,a) \leftarrow q(s,a) + \alpha_t \{ r' + \gamma \max_{a'} q(s',a') - q(s,a) \}$

  $s \leftarrow s'$
- Return q

↑ To ensure convergence of the Q-learning algorithm, one should make sure that each pair (s,a) is visited infinitely many times.

(+ $\sum \alpha_t = +\infty$, $\sum \alpha_t^2 < +\infty$).

V. SCALING UP

V.1. Value Function Approximation.

- So far, given a policy π, a single value of $v_\pi(s)$ or $q_\pi(s,a)$ is stored for each s (or each pair (s,a)).

"look-up" tables

This becomes impractical when the state-space is large.

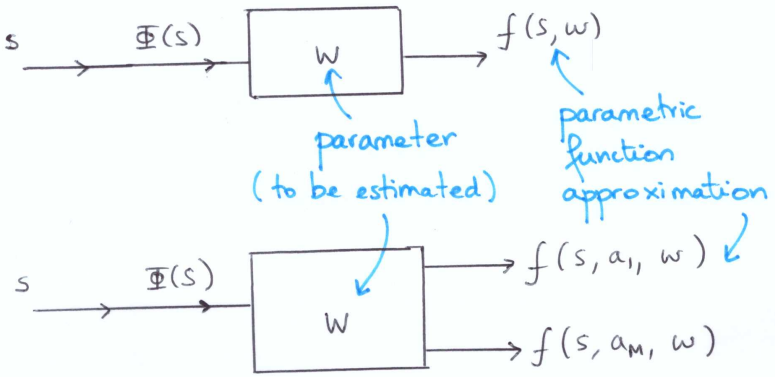
Ex: Go has $\approx 10^{170}$ states.
Backgammon $\approx 10^{20}$ states.

- Idea: Approximate the value function (resp. the action-value function) & generalize from seen states to unseen states

Consider the mapping $\Phi : \mathcal{S} \to \mathbb{R}^d$, with d relatively small.

↑ a "feature map"

$\forall s \in \mathcal{S}$, $\Phi(s)$ represents some compressed knowledge about s.
The low-dimensional feature vector is then used to construct a function $f(s, w) \equiv$ approximation of $v_\pi(s)$.



The problem of estimating a good value of w learning the input (s) – output ($v_\pi(s)$) relationship can be expressed as a supervised learning problem.

- Ex: (i) w = vector of parameters in a linear model

$$f(s,w) = \langle \Phi(s), w \rangle = \Phi(s)^t w$$

Special case: $\Phi : \mathcal{S} \to \mathbb{R}^{|\mathcal{S}|}$, so that Φ is the mapping defined by

$$\Phi(s_j) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow j \quad ; \quad \mathcal{S} = \{s_1, \dots, s_{|\mathcal{S}|}\}$$

Then $f(s_j, w) = w_j$ = value of state s_j.

⇒ Look-up tables are a special case of approximation methods.

(ii) w = set of weights in a neural network.

(iii) Any other learning algorithm, such as decision trees, nearest-neighbours, ... However, models with differentiable properties are usually preferred, as gradient descent algorithms can be used to estimate w.

- To start, suppose that we are in presence of an oracle: for each state s, we know its true label $v_\pi(s)$.

Then given the "training data" (not independent)

$$\{(s_1, v_\pi(s_1)), \dots, (s_T, v_\pi(s_T))\},$$

gathered from experience (≡ MC learning, TD learning setting), a (stochastic) gradient method applied to the (empirical) square loss can be used to iteratively update the parameters:

$$w^{(t+1)} = w^{(t)} - \alpha \nabla_w \left\{ \frac{1}{2} \left(v_\pi(S_t) - f(S_t, w) \right)^2 \right\} \Big|_{w = w^{(t)}}$$

$$= w^{(t)} + \alpha \left(v_\pi(S_t) - f(S_t, w^{(t)}) \right) \nabla_w f(S_t, w) \Big|_{w = w^{(t)}}$$

For a linear model, $\nabla_w f(S_t, w) = \Phi(S_t)$, and the update reduces to:

$$w^{(t+1)} = w^{(t)} + \alpha \left(v_\pi(S_t) - f(S_t, w^{(t)}) \right) \Phi(S_t).$$

- In practice however, there are no oracle / supervisor. We must substitute a target for $v_\pi(s)$, estimated from experience. The most common scenarios are:

  → MC $\quad v_\pi(S_t) \to G_t$ (page 4)

  → TD(0) $\quad v_\pi(S_t) \to R_{t+1} + \gamma f(S_{t+1}, w^{(t)})$

  → TD(λ) $\quad v_\pi(S_t) \to G_t^\lambda$ (page 30)

  ⇒ In the context of MC learning, the training data is $\{(S_1, G_1), (S_2, G_2), \dots, (S_T, G_T)\}$, while in the context of TD(0) learning, we consider $\{(S_1, R_2 + \gamma f(S_2, w)), (S_2, R_3 + \gamma f(S_3, w)), \dots, (S_{T-1}, R_T)\}$

  And cycle through the data / use mini-batches (but randomly)

- Control: These updates can be combined with ε-greedy exploration for policy improvement.

  ↳ Need to consider approximation for the action-value function:

  TD(0) update "on-line"

$$w^{(t+1)} = w^{(t)} + \alpha \left(R_{t+1} + \gamma f(S_{t+1}, A_{t+1}, w^{(t)}) - f(S_t, A_t, w^{(t)}) \right) \times \nabla_w f(S_t, A_t, w) \Big|_{w = w^{(t)}}$$

V.2. Policy Gradient Methods.

- An alternative to the action-value method presented in the previous section is to directly parametrize the policy:

$$\pi_\theta(a \mid s) = \mathbb{P}(A_t = a \mid S_t = a \,;\, \theta)$$

↑ Parameter to be learnt.

↳ Compare with the parametrization of $v_\pi(s) \to f(s, w)$, $q_\pi(s, a) \to f(s, a, w)$

The policy is not represented explicitely, but only implicitely once q_π is estimated.

When parametrizing the policy, we still learn a value function to learn the policy parameter θ; but not to select actions.

- value-based → learnt value function / implicit policy
- policy-based → no value function / learnt policy
- actor-critic → learnt value function / learnt policy.

θ is learnt based on the gradient of some performance measure $J(\theta)$, that we wish to maximize:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)}$$

gradient ascent

Stochastic estimate of the gradient of J evaluated at θ_t: "On average, points in the right direction".

Such algorithms are known as Policy Gradient Methods (whether or not a value function is learnt)

- The policy is usually parametrized using a soft-max representation:

$$\pi_\theta(a|s,\theta) := \frac{e^{h(s,a,\theta)}}{\sum_{a'} e^{h(s,a',\theta)}},$$

where $\rightarrow h(s,a,\theta) = \theta^t \underbrace{\Phi(s,a)}_{\text{feature vector}}$ = linear in features

$\rightarrow h(s,a,\theta)$ = output of a deep neural network ($\rightarrow$ Alpha Go).

- Consider the episodic case: the performance measure $J(\theta)$ is the value of the start state of each episode: $J(\theta) = v_{\pi_\theta}(s_0)$ (s_0: non-random)

We need to compute the gradient of J:

Recall that $v_\pi(s) = \sum_a \pi(a|s)\, q_\pi(s,a)$ (p. 7/8)

$$\Rightarrow \nabla v_\pi(s) = \sum_a \Big(\nabla\pi(a|s)\, q_\pi(s,a) + \pi(a|s)\, \nabla q_\pi(s,a) \Big)$$

$$= \sum_a \Big(\nabla\pi(a|s)\, q_\pi(s,a) + \pi(a|s) \times \nabla\Big(r(s,a) + \underbrace{\gamma}_{=1} \sum_{s'} p(s'|s,a)\, v_\pi(s') \Big) \Big) \quad \text{(p. 8)}$$

$$= \sum_a \Big(\nabla\pi(a|s)\, q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a)\, \nabla v_\pi(s') \Big)$$

("unroll")

$$= \sum_a \Big(\nabla\pi(a|s)\, q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \Big\{ \sum_{a'} \nabla\pi(a'|s')\, q_\pi(s',a') + \pi(a'|s') \sum_{s''} p(s''|s',a')\, \nabla v_\pi(s'') \Big\} \Big)$$

$$= \sum_{s'} \Big(\underbrace{\sum_a \pi(a|s)\, p(s'|s,a)}_{=: p(s \rightarrow s', 1, \pi)} \Big) \Big(\sum_{a'} \nabla\pi(a'|s')\, q_\pi(s',a') \Big)$$

$=: p(s \rightarrow s', 1, \pi)$ = proba of going from state s to state s' in one step, following policy π

Keep unrolling:

terms $\sum_{s'} p(s \rightarrow s', k, \pi) \sum_{a'} \nabla\pi(a'|s')\, q_\pi(s',a')$ appear.

$$\Rightarrow \nabla v_\pi(s) = \sum_{s'} \sum_{k \geq 0} p(s \rightarrow s', k, \pi) \sum_{a'} \nabla\pi(a'|s')\, q_\pi(s',a').$$

Thus

$$\nabla J(\theta) = \sum_s \underbrace{\Big(\sum_{k \geq 0} p(s_0 \rightarrow s, k, \pi) \Big)}_{=: \eta(s)} \sum_a \nabla\pi(a|s)\, q_\pi(s,a).$$

(s_0: initial deterministic state)

$$= \sum_s \eta(s) \sum_a \nabla\pi(a|s)\, q_\pi(s,a)$$

$$= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \Big(\sum_a \nabla\pi(a|s)\, q_\pi(s,a) \Big)$$

$=: \mu(s)$

fraction of time spent in each state s.
μ sums to one (aka the "on-policy distribution")

$$= \sum_{s'} \eta(s') \sum_s \mu(s) \Big(\sum_a \nabla\pi(a|s)\, q_\pi(s,a) \Big)$$

$$\propto \sum_s \mu(s) \Big(\sum_a \nabla\pi(a|s)\, q_\pi(s,a) \Big)$$

(proportional to) (enough for us $\rightarrow$ will be absorbed in the learning rate later)

$$\boxed{\nabla J(\theta) \propto \mathbb{E}_\pi \Big(\sum_a q_\pi(S_t,a)\, \nabla\pi(a|S_t,\theta) \Big)}$$

aka the Policy Gradient Theorem.

replacing s with S_t will allow us to make of samples collected from experience to compute $\nabla J(\theta)$ $\Rightarrow$ we want to replace a with A_t as well.

Next, notice that

$$\nabla \pi(a|s,\theta) = \pi(a|s,\theta)\,\frac{\nabla \pi(a|s,\theta)}{\pi(a|s,a)}$$

$$= \pi(a|s,\theta)\,\nabla \ln \pi(a|s,\theta)$$

Thus

$$\nabla J(\theta) \propto \mathbb{E}_\pi\left(\sum_a \pi(a|S_t,\theta)\, q_\pi(S_t,a)\,\nabla \ln \pi(a|S_t,\theta)\right)$$

$$= \mathbb{E}_\pi\left(q_\pi(S_t,A_t)\,\nabla \ln \pi(A_t|S_t,\theta)\right)$$

$$= \mathbb{E}_\pi\left(G_t\,\nabla \ln \pi(A_t|S_t,\theta)\right)$$

(G_t: return)

We can make use of this expression using a SG algorithm.

⇒ Updates are $\theta_{t+1} = \theta_t + \alpha\, G_t\, \nabla \ln \pi(A_t|S_t,\theta_t)$.

MC policy-gradient control.

(generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(\cdot|\cdot,\theta)$, and for each step of the episode $t = 0, \ldots, T-1$, update $\theta \leftarrow \theta + \alpha\, G\, \nabla \pi(A_t|S_t,\theta)$)

& repeat for each episode.

$G = \sum_{k=t+1}^{T} R_k$

(taking $\gamma = 1$)

This approach can be improved, see e.g. Actor-Critic Methods.