Reinforcement learning

Mathieu Ribatet—Full Professor of Statistics

References

[1] Emma Brunksill. Cs234: Reinforcement learning. `http://web.stanford.edu/class/cs234/index.html`.

[2] David Silver. Reinforcement learning. `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. Adaptive computation and machine learning series. MIT Press, second edition, 2018.

Grading: Reinforcement

1. Go to the Gymnasium website and have a look at the environments (left panel)

   Part I Q–learning // SARSA

   (a) Implement both strategies on one of the following environments: Blackjack or Frozen Lake
   (b) Comment your results

   Part II REINFORCE

   (a) Implement this strategy on one of the following environments: Acrobot or Pendulum
   (b) Comment your results.

Remark. No copy paste of existing code (if any) I will check it and grade 0.

0. Introduction

What is reinforcement learning?

- Reinforcement learning denotes algorithms who learn how to make a sequence of decisions/actions.
- Once an action is taken the algorithm gets rewarded.
- The algorithm hence learns from past experience and obviously aims at maximizing the total reward.

Overall structure: Agent // Environment

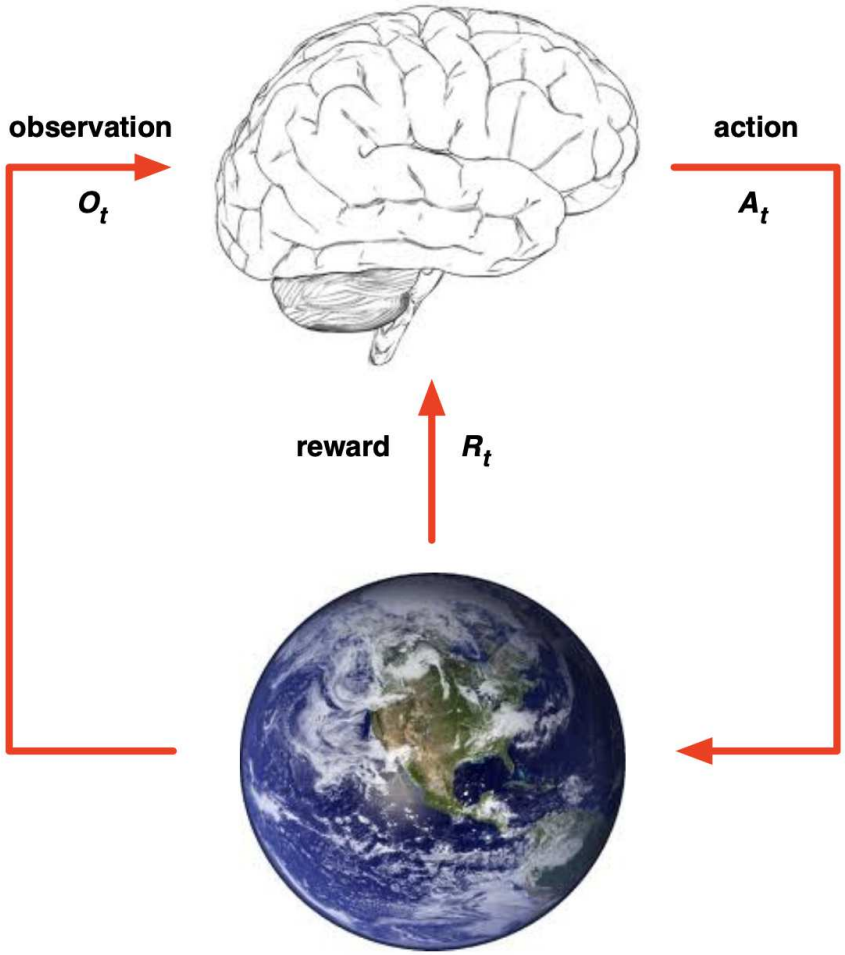


Figure 1: *Schematic diagram of the environment // agent interaction. (Taken from D. Silver)*

- At each time step t the agent:
  - get reward R_t
  - get observation O_t
  - executes action A_t
- At each time step t the environment:
  - get action A_t
  - emits observation O_{t+1}
  - emits reward R_{t+1}.

Specificities of Reinforcement learning

What makes reinforcement learning different from standard machine learning problems?

- It is "half supervised" in the sense that we get rewarded for our actions
- Time dependent, i.e., serial dependence
- Actions impacts future observations
- Deal with censored observations since we do not know what would have happened if we took another decision.

Reinforcement learning agent core

An agent is typically composed of:

- a policy π that controls how the agent takes action;
- value functions v or q that say how good is each state and/or action;
- a model which is a parametric model of how the agent perceives the environement.

Two different objectives

When we make sequential decision making there are typically two different goals:

Reinforcement learning where the environment is initially unknown, the agent interacts with it and improve its policy.

Planning where the environment is known (from a known model) so that the agent take actions w.r.t. that model and improve its policy.

1. Markov Decision Processes

Markov Decision Process

- Markov Decision processes is (almost) the right mathematical object to study in reinforcement learning.

Definition 1. A (finite) Markov Decision Process (MDP) is a tuple $\mathcal{M} = (\mathcal{X}, \mathcal{A}, r, p, \gamma)$ where:

- $\mathcal{X}$ and $\mathcal{A}$ are (finite) sets of states and actions respectively;
- r is a reward function

$$r(x, a) = \mathbb{E}(R_{t+1} \mid X_t = x, A_t = a), \qquad x \in \mathcal{X}, a \in \mathcal{A}.$$

- p is a state transition kernel, given by for any $x' \in \mathcal{X}$,

$$p(x' \mid x, a) = \Pr(X_{t+1} = x' \mid X_t = x, A_t = a), \quad (x, a) \in \mathcal{X} \times \mathcal{A}.$$

- $\gamma \in [0, 1]$ is a discount factor.

Another representation of Markov Decision processes

- Sometimes MDPs are defined using a state–reward transition kernel

$$p(x', r \mid x, a) = \Pr(X_{t+1} = x', R_{t+1} = r \mid X_t = x, A_t = a), \quad (x', r) \in \mathcal{X} \times \mathcal{R}.$$

- But typically it is enough to know the above reward function

$$r(x, a) = \mathbb{E}(R_{t+1} \mid X_t = x, A_t = a).$$

- But if we were willing to use the above representation, clearly we have (supposing $\mathcal{R}$ finite)

$$p(x' \mid x, a) = \sum_{r \in \mathcal{R}} p(x', r \mid x, a)$$

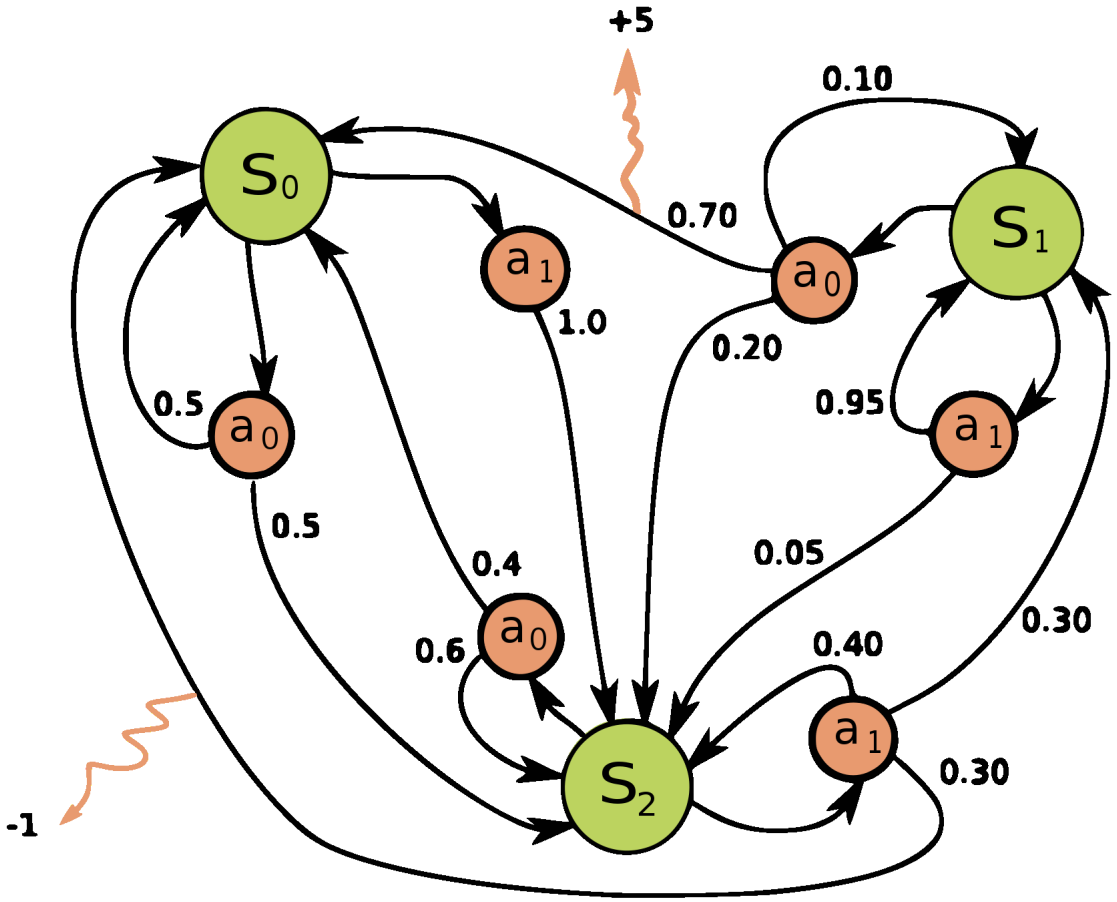


Figure 2: *An example of a Markov decision process with 3 states (S_0, S_1 and S_2), two actions (a_0 and a_1) and two rewards ($+5$ and -1). [Picture taken from wikipedia]*

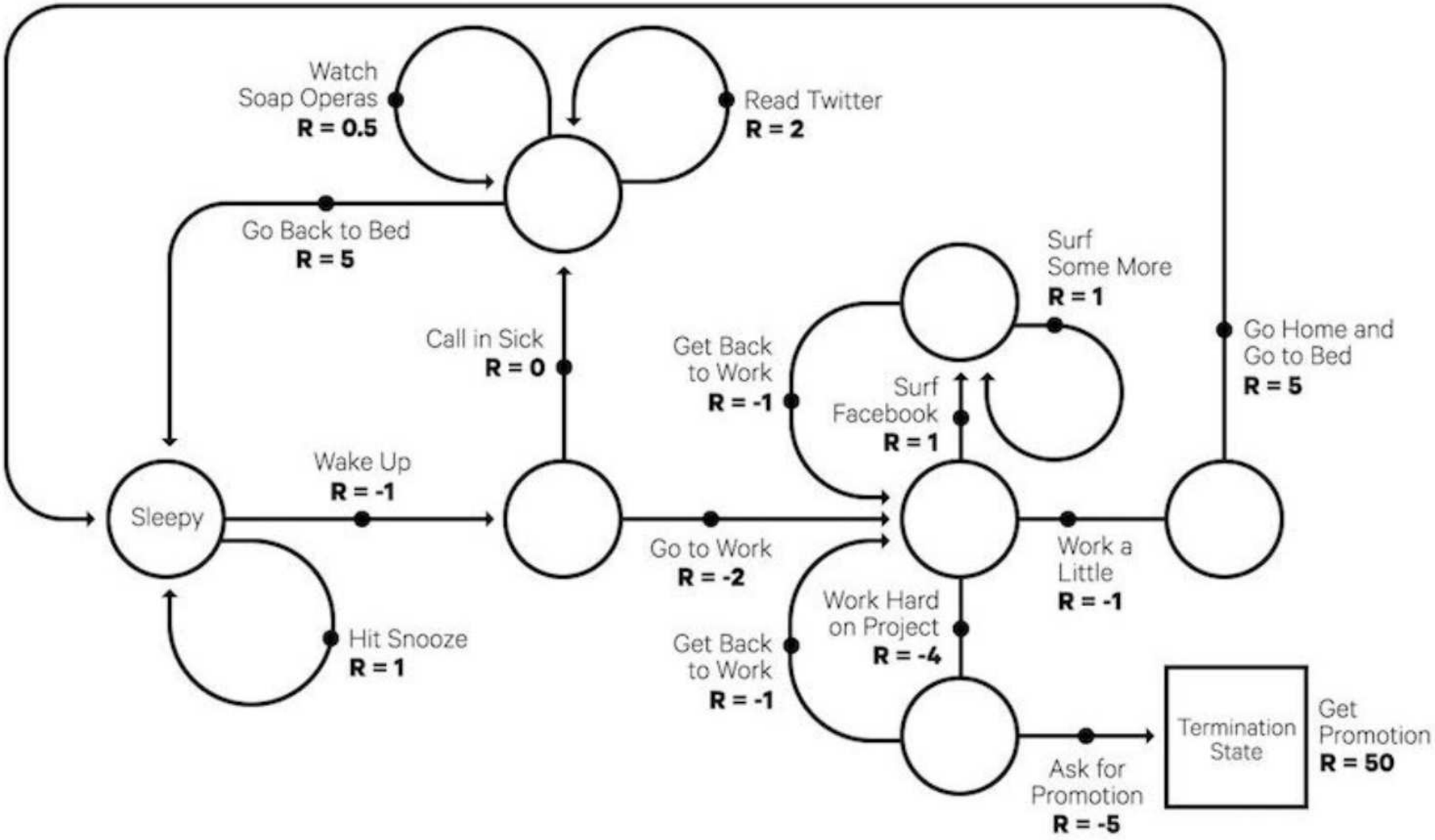
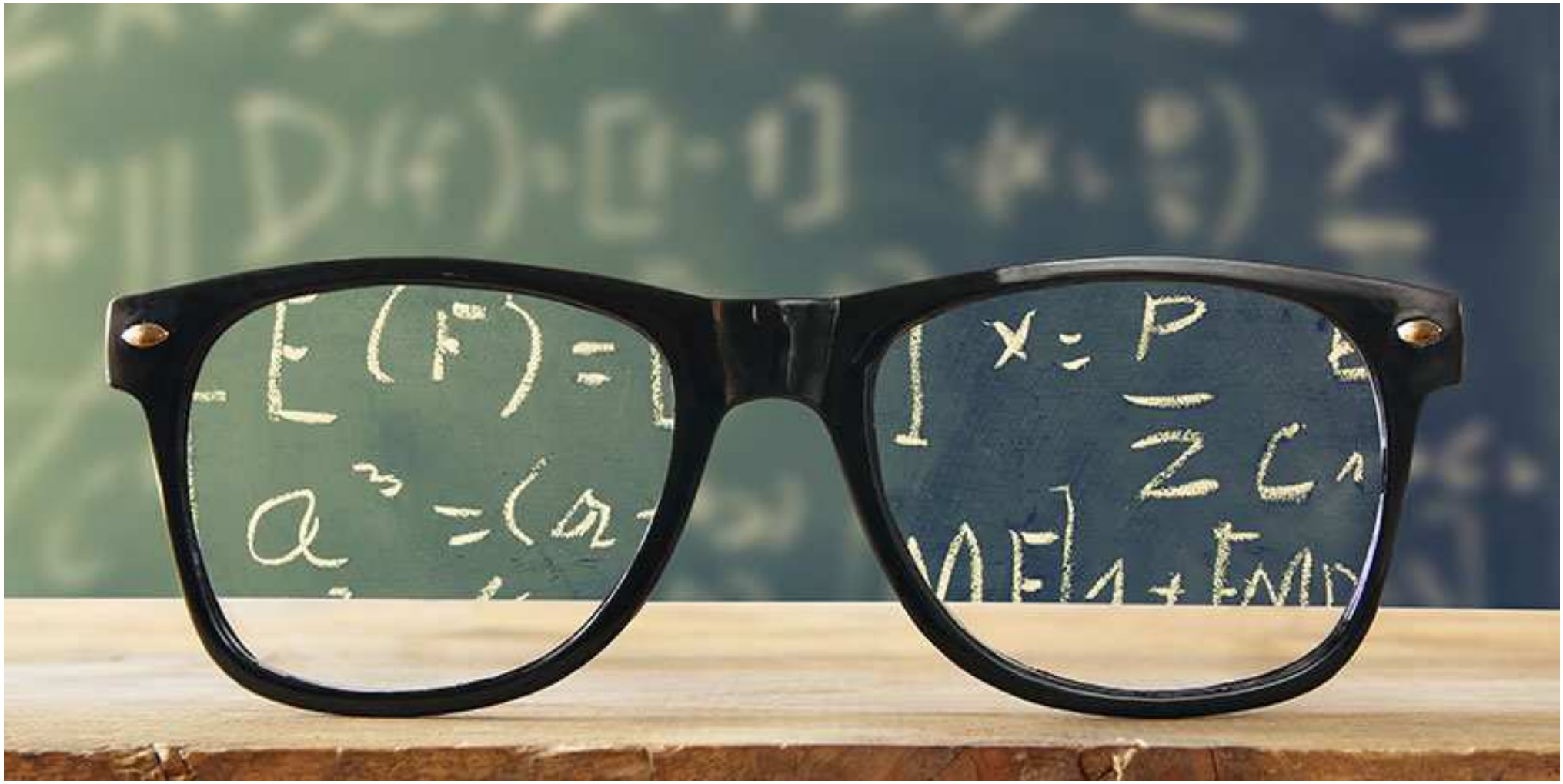


Figure 3: *Another example of a Markov decision process "mimicking a typical day at work". [Picture taken from Random Ant web site]*



Returns of a Markov Decision Process

Definition 2. The (discounted) return of a MDP $\{\mathcal{X}, \mathcal{A}, \mathcal{R}, p, \gamma\}$ is given by

$$G_t = R_{t+1} + \sum_{h \geq 1} \gamma^h R_{t+1+h}, \qquad t \geq 0.$$

- γ close to 0 leads to "myopic" evaluation
- γ close to 1 leads to "far–sighted" evaluation

Returns of a Markov Decision Process

Definition 2. The (discounted) return of a MDP $\{\mathcal{X}, \mathcal{A}, \mathcal{R}, p, \gamma\}$ is given by

$$G_t = R_{t+1} + \sum_{h \geq 1} \gamma^h R_{t+1+h}, \qquad t \geq 0.$$

- γ close to 0 leads to "myopic" evaluation
- γ close to 1 leads to "far–sighted" evaluation

☞ We typically set $\gamma \in [0, 1)$ to ensure that G_t is finite (provided that the rewards are bounded). However we can set $\gamma = 1$ when the state space $\mathcal{X}$ has absorbing/terminal states, i.e., once you're there you're stuck for ever.

Policy

Definition 3. A policy π is a probability distribution function over actions given states, i.e.,

$$\pi(a \mid x) = \Pr(A_t = a \mid X_t = x).$$

- The policy π completely characterizes the behaviour of an agent
- MDP policies depend on the current state only, i.e.,

Stochastic

$$A_t \mid X_t = x \overset{\text{iid}}{\sim} \pi(\cdot \mid x), \qquad \forall t \geq 0.$$

Deterministic

$$a = \pi(x) \qquad \text{(or more rigourously } \pi(\cdot \mid x) = \delta_a(\cdot)\text{)}.$$

Policy

Definition 3. A policy π is a probability distribution function over actions given states, i.e.,

$$\pi(a \mid x) = \Pr(A_t = a \mid X_t = x).$$

- The policy π completely characterizes the behaviour of an agent
- MDP policies depend on the current state only, i.e.,

Stochastic

$$A_t \mid X_t = x \overset{\text{iid}}{\sim} \pi(\cdot \mid x), \qquad \forall t \geq 0.$$

Deterministic

$$a = \pi(x) \qquad \text{(or more rigourously } \pi(\cdot \mid x) = \delta_a(\cdot)\text{)}.$$

☞ Most often, though not invariably, we can focus on deterministic policies.

Value functions

Definition 4. The state–value function v_π of a MDP is the expected return starting from state x and following policy π, i.e.,

$$v_\pi(x) = \mathbb{E}_\pi(G_t \mid X_t = x),$$

where $\mathbb{E}_\pi$ denotes the expectation when the agent follows policy π.

Definition 5. The state–action–value function q_π of a MDP is the expected return starting from state x, taking action a, and thereafter following policy π, i.e.,

$$q_\pi(x, a) = \mathbb{E}_\pi(G_t \mid X_t = x, A_t = a).$$

Exercise 1. Show that, for all $x \in \mathcal{X}$, $v_\pi(x) = \sum_{a \in \mathcal{A}} q(x, a)\pi(a \mid x)$.

Proof. Hint: $\mathbb{E}_Y\{\mathbb{E}_X(X \mid Y)\} = \mathbb{E}(X)$ *and let* $Z = G_t \mid X_t$. □

Value functions

Definition 4. The state–value function v_π of a MDP is the expected return starting from state x and following policy π, i.e.,

$$v_\pi(x) = \mathbb{E}_\pi(G_t \mid X_t = x),$$

where $\mathbb{E}_\pi$ denotes the expectation when the agent follows policy π.

Definition 5. The state–action–value function q_π of a MDP is the expected return starting from state x, taking action a, and thereafter following policy π, i.e.,

$$q_\pi(x, a) = \mathbb{E}_\pi(G_t \mid X_t = x, A_t = a).$$

Exercise 1. Show that, for all $x \in \mathcal{X}$, $v_\pi(x) = \sum_{a \in \mathcal{A}} q(x, a)\pi(a \mid x)$.

Proof. Hint: $\mathbb{E}_Y\{\mathbb{E}_X(X \mid Y)\} = \mathbb{E}(X)$ *and let* $Z = G_t \mid X_t$. □

☞ If $x \in \mathcal{X}$ is a terminal state then $v_\pi(x) = q_\pi(x, a) = 0$, $a \in \mathcal{A}$.

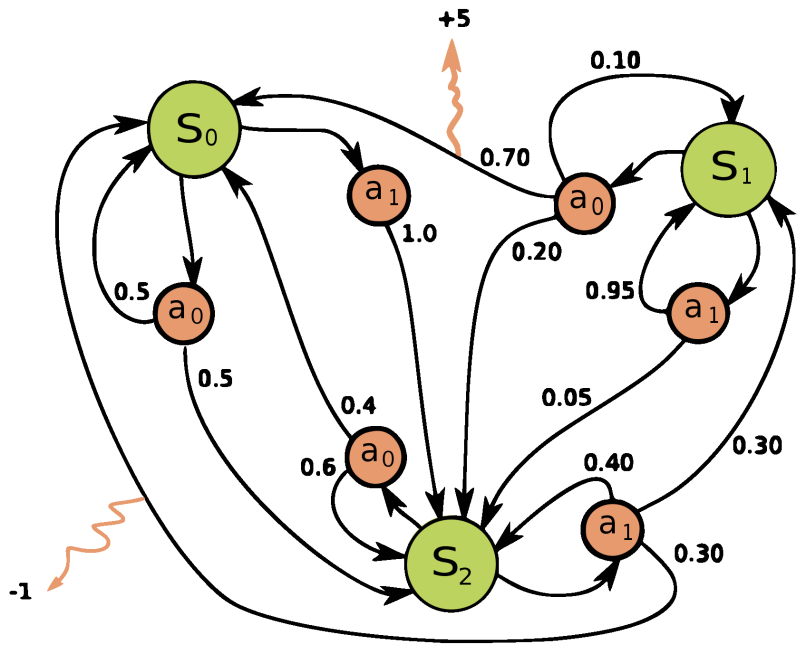


Figure 4: *An example of a Markov decision process with 3 states (S_0, S_1 and S_2), two actions (a_0 and a_1) and two rewards ($+5$ and -1). [Picture taken from wikipedia]*

Following the random policy

$$\pi \sim U\{a_0, a_1\},$$

we have

$$\mathbf{v} \approx \begin{cases} (\infty, \infty, \infty), & \gamma = 1.00 \\ (1.47, 4.55, 1.69), & \gamma = 0.90 \\ (0.31, 3.09, 0.45), & \gamma = 0.75 \\ (0.02, 2.38, 0.04), & \gamma = 0.50 \\ (-0.02, 2.01, -0.09), & \gamma = 0.25 \\ (-0.01, 1.84, -0.13), & \gamma = 0.10 \\ (0.00, 1.75, -0.15), & \gamma = 0.00 \end{cases}$$

Bellman expectation equation

$$v_\pi(x) = \mathbb{E}_\pi \left\{ R_{t+1} + \gamma v_\pi(X_{t+1}) \mid X_t = x \right\}, \qquad x \in \mathcal{X}.$$

Proof.

$$\begin{aligned}
v_\pi(x) &= \mathbb{E}_\pi(G_t \mid X_t = x) \\
&= \mathbb{E}_\pi(R_{t+1} + \gamma G_{t+1} \mid X_t = x) \\
&= \mathbb{E}_\pi(R_{t+1} + \gamma \mathbb{E}(G_{t+1} \mid X_{t+1}, X_t = x) \mid X_t = x) \\
&= \mathbb{E}_\pi(R_{t+1} + \gamma \mathbb{E}(G_{t+1} \mid X_{t+1}) \mid X_t = x) \\
&= \mathbb{E}_\pi(R_{t+1} + \gamma v_\pi(X_{t+1}) \mid X_t = x)
\end{aligned}$$

□

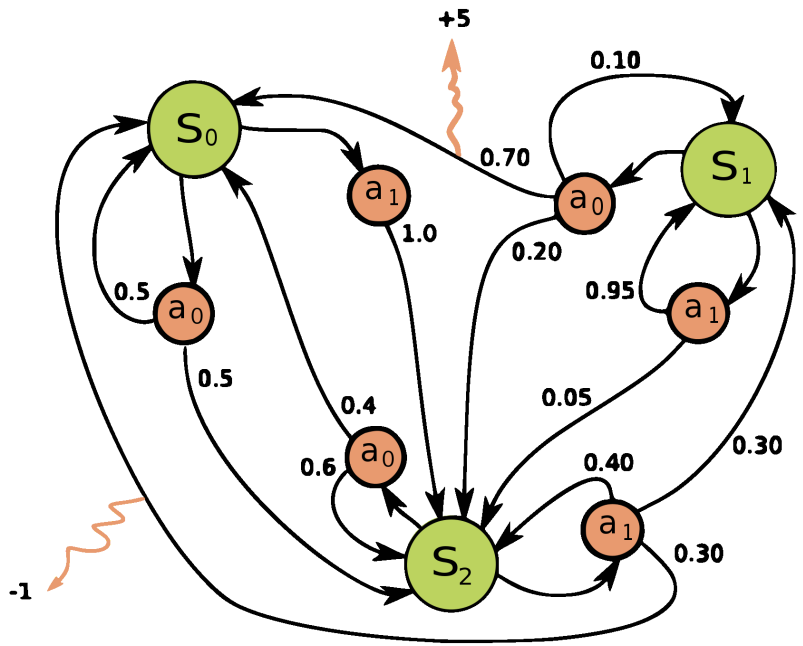


Figure 5: *An example of a Markov decision process with 3 states (S_0, S_1 and S_2), two actions (a_0 and a_1) and two rewards ($+5$ and -1). [Picture taken from wikipedia]*

Exercise 2. Consider the Markov decision process plotted on the left and the random policy $\pi = U\{a_0, a_1\}$. Show that, when the discount factor $\gamma = 0$, the value function $\mathbf{v}_\pi = (v(S_0), v(S_1), v(S_2))$ is indeed

$$\mathbf{v}_\pi = (0, 1.75, -0.15).$$

Bellman expectation equation (2)

$$q_\pi(x,a) = \mathbb{E}_\pi \left\{ R_{t+1} + \gamma q_\pi(X_{t+1}, A_{t+1}) \mid X_t = x, A_t = a \right\}, \qquad (x,a) \in \mathcal{X} \times \mathcal{A}.$$

Proof. Similar idea $\Rightarrow$ Homework. □

Bellman expectation equation (3)

- Recall the first version of the Bellman expectation equation

$$v_\pi(x) = \mathbb{E}_\pi \left\{ R_{t+1} + \gamma v_\pi(X_{t+1}) \mid X_t = x \right\}, \qquad x \in \mathcal{X}.$$

- Using matrix notation, it now writes $V_\pi = R_\pi + \gamma P_\pi V_\pi$ where

$$R_\pi = \{r_\pi(x) \colon x \in \mathcal{X}\}, \qquad P_\pi = \{p_\pi(x, x') \colon x, x' \in \mathcal{X}\}$$
$$r_\pi(x) = \sum_{a \in \mathcal{A}} \pi(a \mid x) r(x, a), \qquad p_\pi(x, x') = \sum_{a \in \mathcal{A}} \pi(a \mid x) p(x' \mid x, a).$$

- So that, provided it exists, we have $V_\pi = (\mathsf{Id} - \gamma P_\pi)^{-1} R_\pi$.

Bellman expectation equation (3)

- Recall the first version of the Bellman expectation equation

$$v_\pi(x) = \mathbb{E}_\pi \left\{ R_{t+1} + \gamma v_\pi(X_{t+1}) \mid X_t = x \right\}, \qquad x \in \mathcal{X}.$$

- Using matrix notation, it now writes $V_\pi = R_\pi + \gamma P_\pi V_\pi$ where

$$R_\pi = \{r_\pi(x) \colon x \in \mathcal{X}\}, \qquad P_\pi = \{p_\pi(x, x') \colon x, x' \in \mathcal{X}\}$$
$$r_\pi(x) = \sum_{a \in \mathcal{A}} \pi(a \mid x) r(x, a), \qquad p_\pi(x, x') = \sum_{a \in \mathcal{A}} \pi(a \mid x) p(x' \mid x, a).$$

- So that, provided it exists, we have $V_\pi = (\mathsf{Id} - \gamma P_\pi)^{-1} R_\pi$.

☞ If we use deterministic policies, we have

$$r_\pi(x) = r(x, \pi(x)), \qquad p_\pi(x, x') = p(x' \mid x, \pi(x)).$$

How to choose policies?

- Given two policies π_1 and π_2, which one should we prefer?
- Clearly it is better to follow a policy that achieves a lot of reward over the long run, i.e.,

$$\text{maximize } v_\pi(x) = \mathbb{E}_\pi(G_t \mid X_t = x)$$

Definition 6. We say that π_1 is better than π_2 ($\pi_1 \geq \pi_2$) if

$$v_{\pi_1}(x) \geq v_{\pi_2}(x), \qquad x \in \mathcal{X},$$

and it thus define a partial ordering on policies.

Optimal value function

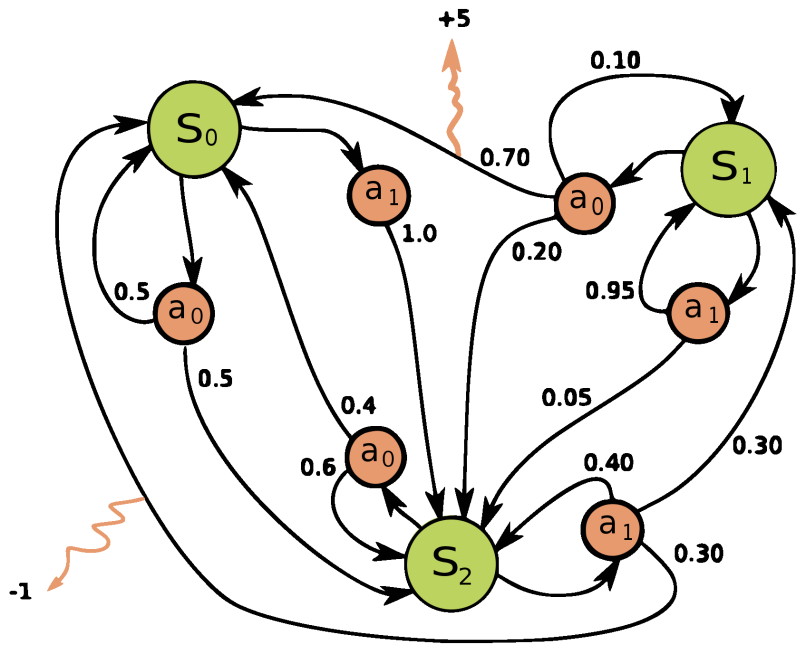
Definition 7. The optimal state–value function v_* is the maximum state–value function over all policies, i.e.,

$$v_*(x) = \max_\pi v_\pi(x), \qquad x \in \mathcal{X}.$$

The optimal action–value function q_* is the maximum action–value function over all policies, i.e.,

$$q_*(x,a) = \max_\pi q_\pi(x,a), \qquad (x,a) \in \mathcal{X} \times \mathcal{A}.$$

☞ Optimal value functions characterize the best achievable performance.



The optimal value function is

$$\mathbf{v}_* \approx \begin{cases} (\infty, \infty, \infty), & \gamma = 1.00 \\ (3.79, 7.30, 4.21), & \gamma = 0.90 \\ (3.16, 6.67, 3.46), & \gamma = 0.75 \\ (2.11, 5.61, 2.21), & \gamma = 0.50 \\ (1.05, 4.56, 1.01), & \gamma = 0.25 \\ (0.42, 3.92, 0.40), & \gamma = 0.10 \\ (0.00, 3.50, 0.00), & \gamma = 0.00 \end{cases}$$

Figure 6: *An example of a Markov decision process with 3 states (S_1, S_2 and S_3), two actions (a_0 and a_1) and two rewards ($+5$ and -1). [Picture taken from wikipedia]*

Theorem 1. *For any Markov Decision Process,*

- *there exists at least one optimal policy π_* such that $\pi_* \geq \pi$ for all policies π;*
- *there is always a deterministic optimal policy (not necessarily the best one, i.e., random ones)*
- *all optimal policies achieve the optimal value function, i.e., $v_{\pi_*}(x) = v_*(x)$ for all $x \in \mathcal{X}$ (by definition);*
- *all optimal policies achieve the optimal action–value function, i.e., $q_{\pi_*}(x, a) = q_*(x, a)$ for all $(x, a) \in \mathcal{X} \times \mathcal{A}$.*

Proof. Admitted (but finiteness of $\mathcal{X}$ and $\mathcal{A}$ clearly helps here!) □

Deriving the optimal deterministic policy

- Recall that (by definition)

$$v_*(x) = \sum_{a \in \mathcal{A}} \pi(a \mid x) q_*(x, a).$$

- Since there is always a deterministic optimal policy we can write

$$\pi_*(\cdot \mid x) = \delta_{a_x}(\cdot), \quad \text{for some } a_x \in \mathcal{A}.$$

- Hence for all $x \in \mathcal{X}$ we have

$$v_*(x) = q_*(x, a_x) = \max_{a \in \mathcal{X}} q_*(x, a)$$

where the last equality used the fact that v_* is necessarily optimal.

Deriving the optimal deterministic policy

- Recall that (by definition)

$$v_*(x) = \sum_{a \in \mathcal{A}} \pi(a \mid x) q_*(x, a).$$

- Since there is always a deterministic optimal policy we can write

$$\pi_*(\cdot \mid x) = \delta_{a_x}(\cdot), \quad \text{for some } a_x \in \mathcal{A}.$$

- Hence for all $x \in \mathcal{X}$ we have

$$v_*(x) = q_*(x, a_x) = \max_{a \in \mathcal{X}} q_*(x, a)$$

  where the last equality used the fact that v_* is necessarily optimal.
- As an aside an optimal (deterministic) policy is always given by

$$\pi_*(a \mid x) = \begin{cases} 1, & \text{if } a = \arg\max\limits_{a \in \mathcal{A}} q_*(x, a) \\ 0, & \text{otherwise.} \end{cases}$$

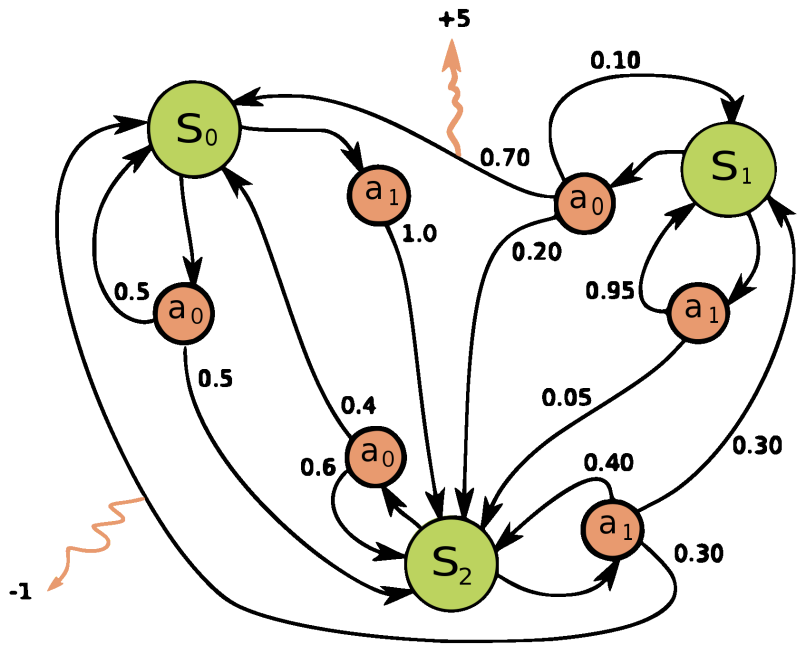


Figure 7: *An example of a Markov decision process with 3 states (S_1, S_2 and S_3), two actions (a_0 and a_1) and two rewards ($+5$ and -1). [Picture taken from wikipedia]*

The optimal (deterministic) policy is

$$\pi_* \approx \begin{cases} \text{not defined}, & \gamma = 1.00 \\ (a_1, a_0, a_1), & \gamma = 0.90 \\ (a_1, a_0, a_1), & \gamma = 0.75 \\ (a_1, a_0, a_1), & \gamma = 0.50 \\ (a_1, a_0, a_0), & \gamma = 0.25 \\ (a_1, a_0, a_0), & \gamma = 0.10 \\ (\{a_0, a_1\}, a_0, a_0), & \gamma = 0.00 \end{cases}$$

Bellman optimality equation (for v_*)

$$v_*(x) = \max_{a \in \mathcal{A}} \mathbb{E}\left\{R_{t+1} + \gamma v_*(X_{t+1}) \mid X_t = x, A_t = a\right\}, \qquad x \in \mathcal{X}.$$

Proof. Remember that $v_*(x) = \max_{a \in \mathcal{A}} q_{\pi_*}(x, a)$. Hence we get:

$$\begin{aligned} v_*(x) &= \max_a q_{\pi_*}(x, a) \\ &= \max_a \mathbb{E}_{\pi_*}\left(G_t \mid X_t = x, A_t = a\right) \\ &= \max_a \mathbb{E}_{\pi_*}\left(R_{t+1} + \gamma G_{t+1} \mid X_t = x, A_t = a\right) \\ &= \max_a \mathbb{E}\left(R_{t+1} + \gamma v_*(X_{t+1}) \mid X_t = x, A_t = a\right) \end{aligned}$$

□

Bellman optimality equation (for v_*)

$$v_*(x) = \max_{a \in \mathcal{A}} \mathbb{E}\left\{R_{t+1} + \gamma v_*(X_{t+1}) \mid X_t = x, A_t = a\right\}, \qquad x \in \mathcal{X}.$$

Proof. Remember that $v_*(x) = \max_{a \in \mathcal{A}} q_{\pi_*}(x, a)$. Hence we get:

$$\begin{aligned} v_*(x) &= \max_a q_{\pi_*}(x, a) \\ &= \max_a \mathbb{E}_{\pi_*}\left(G_t \mid X_t = x, A_t = a\right) \\ &= \max_a \mathbb{E}_{\pi_*}\left(R_{t+1} + \gamma G_{t+1} \mid X_t = x, A_t = a\right) \\ &= \max_a \mathbb{E}\left(R_{t+1} + \gamma v_*(X_{t+1}) \mid X_t = x, A_t = a\right) \end{aligned}$$

□

The Bellman optimality equation is not linear (due to the $\max$).

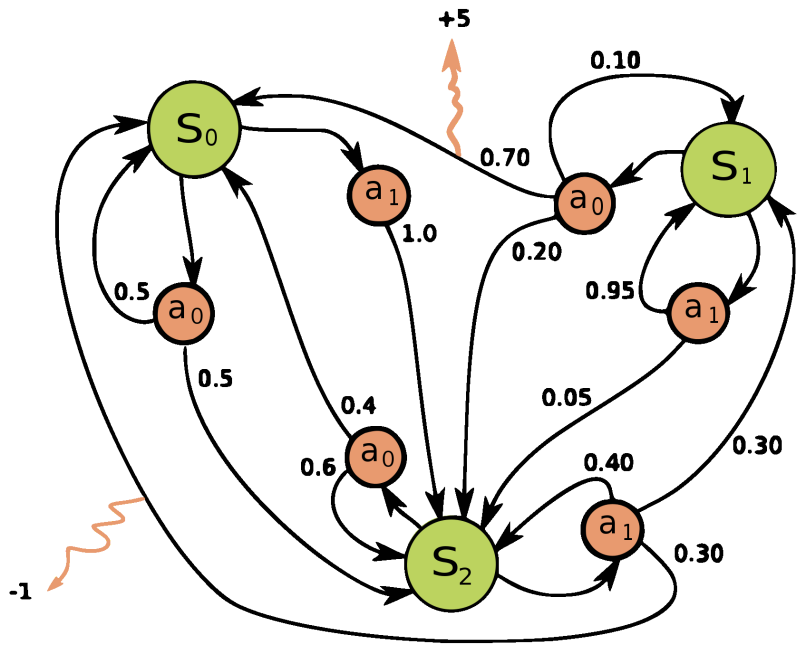


Figure 8: *An example of a Markov decision process with 3 states (S_1, S_2 and S_3), two actions (a_0 and a_1) and two rewards ($+5$ and -1). [Picture taken from wikipedia]*

Exercise 3. Consider the Markov decision process plotted on the left. Show that, when the discount factor $\gamma = 0$, the optimal value function $\mathbf{v}_* = (v_*(S_0), v_*(S_1), v_*(S_2))$ is indeed

$$\mathbf{v}_* = (0, 3.5, 0).$$

Bellman optimality equation (for q_*)

Similarly to the previous optimal equation, we can get the Bellman optimality equation for the action–value function q_*, i.e.,

$$q_*(x,a) = \mathbb{E}\left\{ R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_*(X_{t+1}, a') \mid X_t = x, A_t = a \right\}.$$

Proof. Essentially the same. □

- Bellman optimality equation are non linear (essentially because of the $\max$)
- There is (almost always) no closed form solution
- To get solution we need to use numerical methods such as:
  - Iterative Policy Evaluation
  - Value Iteration
  - Value iteration

2. Dynamic programming

What is dynamic programming?

- Dynamic programming most often refers to algorithm strategies to optimization problem.
- Broadly speaking, it is mainly based on the divide/conquer paradigm.
- Although dynamic programming has been used successfully in many situations, keep in mind that it will not scale well for very large problems
- So why studying dynamic programming?
  - general knowledge
  - reinforcement learning strategies has connections with it

- What we are going to see in the next few slides is a 2 step procedure to get an estimate of a deterministic optimal policy.
- This 2 step procedure is very simple and consists in

  Step 1 Estimate the value function v_π: the iterative policy evaluation step
  Step 2 Improve the current policy π: the policy improvement step

- Steps 1–2 are repeated in turns: it is the policy iteration algorithm.

Step 1: Policy evaluation

- Let π be a policy. Recall that its Bellman expectation equation is

$$v_\pi(x) = \mathbb{E}\left\{R_{t+1} + \gamma v_\pi(X_{t+1}) \mid X_t = x\right\}, \quad x \in \mathcal{X}.$$

- Suppose the dynamic is completely known, i.e., reward function and state transition kernel are known. How can we compute $v_\pi(\cdot)$?
- Essentially we use a fixed point strategy, i.e., for some arbitrary initial value function v_0, define a sequence $\{v_k(\cdot) : k \geq 0\}$ where

$$v_{k+1}(x) = \mathbb{E}\left\{R_{t+1} + \gamma v_k(X_{t+1}) \mid X_t = x\right\}, \quad x \in \mathcal{X}.$$

- Convergence $v_k \to v_\pi$ is guaranteed as long as the update is a contraction—see later.

Iterative policy evaluation

Algorithm 1: Iterative policy evaluation to estimate v_π.

input : The policy π to be evaluated, an initial value function v and a tolerance value $\varepsilon > 0$.

output: An estimate v of v_π.

```
/* To enter the while loop the first time                    */
```

$\Delta \leftarrow 2\varepsilon$;

while $\Delta > \varepsilon$ **do**

  Backup the current value function $v_{\text{old}} \leftarrow v$;

  for $x \in \mathcal{X}$ **do**

    $v(x) \leftarrow \mathbb{E}\{R_{t+1} + \gamma v_{\text{old}}(X_{t+1}) \mid X_t = x\}$;

  $\Delta \leftarrow \|v_{\text{old}}(x) - v(x)\|_\infty$;

Return $\{v(x) \colon x \in \mathcal{X}\}$;

Step 2: Policy improvement (Be greedy)

- Consider a deterministic policy π, i.e., $\pi(x) = a$.
- We can improve π by acting greedily, e.g.,

$$\pi'(x) = \arg\max_{a \in \mathcal{A}} q_\pi(x, a), \qquad x \in \mathcal{X}.$$

- We get a better policy since

$$\begin{aligned} v_\pi(x) &= q_\pi(x, \pi(x)) \leq q_\pi(x, \pi'(x)) = \mathbb{E}_{\pi'}\{R_{t+1} + \gamma v_\pi(X_{t+1}) \mid X_t = x\} \\ &\leq \mathbb{E}_{\pi'}\{R_{t+1} + \gamma q_\pi(X_{t+1}, \pi'(X_{t+1})) \mid X_t = x\} \\ &\leq \mathbb{E}_{\pi'}\{R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(X_{t+2}, \pi'(X_{t+2})) \mid X_t = x\} \\ &\leq \mathbb{E}_{\pi'}\{R_{t+1} + \gamma R_{t+2} + \cdots \mid X_t = x\} \\ &= v_{\pi'}(x). \end{aligned}$$

It works!

- What happens when our greedy improvement stops improving the policy?

It works!

- What happens when our greedy improvement stops improving the policy?
- In such situations, one may have a new optimal policy π' such that $\pi'(x) = \pi(x)$, $x \in \mathcal{X}$.
- However since π' is optimal and identical to π we have

$$q_\pi(x, \pi'(x)) = \max_{a \in \mathcal{A}} q_\pi(x, a) = q_\pi(x, \pi(x)) = v_\pi(x) \qquad \texttt{(deterministic)}$$

- We have just stated the Bellman optimality equation!

$$v_\pi(x) = \max_{a \in \mathcal{A}} q_\pi(x, a),$$

It works!

- What happens when our greedy improvement stops improving the policy?
- In such situations, one may have a new optimal policy π' such that $\pi'(x) = \pi(x)$, $x \in \mathcal{X}$.
- However since π' is optimal and identical to π we have

$$q_\pi(x, \pi'(x)) = \max_{a \in \mathcal{A}} q_\pi(x, a) = q_\pi(x, \pi(x)) = v_\pi(x) \qquad \texttt{(deterministic)}$$

- We have just stated the Bellman optimality equation!

$$v_\pi(x) = \max_{a \in \mathcal{A}} q_\pi(x, a),$$

- Hence π is a (deterministic) optimal policy.

Algorithm 2: Policy iteration.

input : A tolerance value $\varepsilon > 0$ and initial (arbitrary) policy π and value function v.
output: Estimates π and v of π_* and v_* respectively.

$flag \leftarrow TRUE$;
while $flag$ **do**
    /* Policy evaluation */
    $\Delta \leftarrow 2\varepsilon$;
    while $\Delta > \varepsilon$ **do**
        Backup the current value function $v_{\text{old}} \leftarrow v$;
        for $x \in \mathcal{X}$ **do**
            $v(x) \leftarrow \mathbb{E}\{R_{t+1} + \gamma v_{\text{old}}(X_{t+1}) \mid X_t = x\}$;
        $\Delta \leftarrow \|v_{\text{old}}(x) - v(x)\|_\infty$;
    /* Policy improvement */
    $\pi_{\text{old}} \leftarrow \pi(x)$;
    for $x \in \mathcal{X}$ **do**
        $\pi(x) \leftarrow \arg\max_{a \in \mathcal{A}} \mathbb{E}\{R_{t+1} + \gamma v(X_{t+1}) \mid X_t = x, A_t = a\}$;
    $flag \leftarrow 1_{\{\pi \neq \pi_{\text{old}}\}}$;
Return $\{(v(x), \pi(x)) : x \in \mathcal{X}\}$;

- The policy iteration algorithm does not scale well
- Indeed inside the outer `while` loop, we have a policy evalutation step which is typically CPU demanding.
- One can speed up things by just updating the policy π from a single pass over the state $x \in \mathcal{X}$.
- Such a procedure is known as value iteration.

Algorithm 3: Value iteration.

input : A tolerance value $\varepsilon > 0$ and initial (arbitrary) value function v.
output: An estimate π of π_*.

$\Delta \leftarrow 2\varepsilon$;
while $\Delta > \varepsilon$ **do**
    Backup the value function, $v_{\text{old}} \leftarrow v$;
    for $x \in \mathcal{X}$ **do**
        $v(x) \leftarrow \max_a \mathbb{E}\{R_{t+1} + \gamma V_{\text{old}}(X_{t+1}) \mid X_t = x, A_t = a\}$;
    $\Delta \leftarrow |v_{\text{old}}(x) - v(x)|_\infty$;
Return the (deterministic) policy π given by

$$\pi(x) = \arg\max_{a \in \mathcal{A}} \mathbb{E}\{R_{t+1} + \gamma v(X_{t+1}) \mid X_t = x, A_t = a\}.$$

Contraction

- Consider the Bellman expectation updating stage, i.e.,

$$v'(x) = \mathbb{E}_\pi(R_{t+1} + \gamma v(X_{t+1}) \mid X_t = x),$$

which give using vectorial notations

$$V' = R_\pi + \gamma P_\pi V.$$

- We now show that the operator T_π given by $V' = T_\pi(V)$ is a contraction.

$$\begin{aligned}
\|T_\pi(V_1) - T_\pi(V_2)\|_\infty &= \|R_\pi + \gamma P_\pi V_1 - R_\pi + \gamma P_\pi V_2\|_\infty \\
&= \gamma \|P_\pi(V_1 - V_2)\|_\infty \\
&\leq \gamma \|V_1 - V_2\|_\infty \quad P_\pi \text{ is a stochastic matrix,}
\end{aligned}$$

and we thus have a contraction as long as $0 \leq \gamma < 1$.

Convergence

Let V_π be the value function for policy π and consider the sequence $V_{k+1} = T(V_k)$, $k \geq 0$, with V_0 an arbitrary initial value function. We have

$$\begin{aligned}\|V_\pi - V_k\|_\infty &= \|T(V_\pi) - T(V_{k-1})\|_\infty \leq \gamma \|V_\pi - V_{k-1}\|_\infty \\ &\leq \gamma^k \|V_\pi - V_0\|_\infty \\ &\longrightarrow 0, \qquad k \to \infty,\end{aligned}$$

provided that $\|V_\pi - V_0\|\infty$ and $\gamma \in [0, 1)$.
In addition note that

$$\begin{aligned}T^k V_0 &\stackrel{\text{def}}{=} T \circ \cdots \circ T(V_0) = T^{k-1}(R_\pi + \gamma P_\pi V_0) \\ &= R_\pi + \gamma P_\pi R_\pi + \cdots + \gamma^k P_\pi^k R_\pi + \gamma^k P_\pi^k V_0 \\ &\longrightarrow (\mathsf{Id} + \gamma P_\pi + \gamma^2 P_\pi^2 + \cdots) \\ &= (\mathsf{Id} - \gamma P_\pi)^{-1} R_\pi = V_\pi.\end{aligned}$$

3. Model free prediction and control

- So far we learnt how to:
  - make prediction for Markov decision processes, i.e., what will be the (expected) return.
  - control Markov decision processes, i.e., which policy should I follow?
- There is however one (big) limitation...

- So far we learnt how to:
  - make prediction for Markov decision processes, i.e., what will be the (expected) return.
  - control Markov decision processes, i.e., which policy should I follow?
- There is however one (big) limitation. . . the state transition kernel $p(x' \mid x, a)$ and the reward function $r(x, a)$ were supposed to be known.

- So far we learnt how to:
  - make prediction for Markov decision processes, i.e., what will be the (expected) return.
  - control Markov decision processes, i.e., which policy should I follow?
- There is however one (big) limitation. . . the state transition kernel $p(x' \mid x, a)$ and the reward function $r(x, a)$ were supposed to be known.
- We will now see what to do in such cases using two different approaches:
  - Monte Carlo learning
  - Temporal difference learning (TD learning)
- We will cover these strategies in turn.

Monte Carlo learning

- The overall idea in Monte Carlo is to use the law of large numbers

$$\frac{1}{n}\sum_{i=1}^{n} h(X_i) \overset{\text{proba}}{\longrightarrow} \mathbb{E}\{h(X)\}, \qquad n \to \infty,$$

  where $X_1, X_2, \ldots \overset{\text{iid}}{\sim} X$ and $\mathbb{E}\{h(X)\} < \infty$.
- To apply such approach for Markov decision processes we need thoses processes to have terminal/absorbing states to ensure we can have independent copies.
- Hence we suppose we dispose of the following sequences of episodes

$$\{(X_t^{(i)}, A_t^{(i)}, R_{t+1}^{(i)}) \colon t = 0, \ldots, T_i\}_{i=1,\ldots,n}.$$

Monte Carlo Policy evaluation

We would like to estimate $v_\pi(x) = \mathbb{E}_\pi(G_t \mid X_t = x)$.

Monte Carlo Policy evaluation

We would like to estimate $v_\pi(x) = \mathbb{E}_\pi(G_t \mid X_t = x)$.

First visit Estimator For each $x \in \mathcal{X}$

$$\hat{v}_\pi(x) = \frac{\sum_{i=1}^n G^{(i)}_{T_i(x)}}{\sum_{i=1}^n 1_{\{T_i(x) \leq T_i\}}}, \qquad T_i(x) = \min\{t \geq 0 \colon X_t^{(i)} = x\}.$$

Every visit esimator For each $x \in \mathcal{X}$

$$\tilde{v}_\pi(x) = \frac{\sum_{i=1}^n \sum_{t=0}^{T_i} G_t^{(i)} 1_{\{X_t^{(i)} = x\}}}{\sum_{i=1}^n \sum_{t=0}^{T_i} 1_{\{X_t^{(i)} = x\}}}.$$

Monte Carlo Policy evaluation

We would like to estimate $v_\pi(x) = \mathbb{E}_\pi(G_t \mid X_t = x)$.

First visit Estimator For each $x \in \mathcal{X}$

$$\hat{v}_\pi(x) = \frac{\sum_{i=1}^n G^{(i)}_{T_i(x)}}{\sum_{i=1}^n 1_{\{T_i(x) \leq T_i\}}}, \qquad T_i(x) = \min\{t \geq 0 \colon X_t^{(i)} = x\}.$$

Every visit esimator For each $x \in \mathcal{X}$

$$\tilde{v}_\pi(x) = \frac{\sum_{i=1}^n \sum_{t=0}^{T_i} G_t^{(i)} 1_{\{X_t^{(i)} = x\}}}{\sum_{i=1}^n \sum_{t=0}^{T_i} 1_{\{X_t^{(i)} = x\}}}.$$

The first visit estimator was first used as a simple "iid case" but the every visit estimator is now preferred (proof of convergence is more difficult).

As an aside

- Note that

$$\bar{X}_{1:n} = \frac{1}{n}\left(X_n + (n-1)\bar{X}_{1:(n-1)}\right) = \left(1 - \frac{1}{n}\right)\bar{X}_{1:(n-1)} + \frac{1}{n}X_n.$$

- Therefore our previous Monte–Carlo estimator, as an incremental mean, can be written as

$$N(X_t) \leftarrow N(X_t) + 1$$

$$\hat{v}_\pi(X_t) \leftarrow (1-\alpha)\hat{v}_\pi(X_t) + \alpha G_t, \qquad \alpha = \frac{1}{N(X_t)}.$$

As an aside

- Note that

$$\bar{X}_{1:n} = \frac{1}{n}\left(X_n + (n-1)\bar{X}_{1:(n-1)}\right) = \left(1 - \frac{1}{n}\right)\bar{X}_{1:(n-1)} + \frac{1}{n}X_n.$$

- Therefore our previous Monte–Carlo estimator, as an incremental mean, can be written as

$$N(X_t) \leftarrow N(X_t) + 1$$

$$\hat{v}_\pi(X_t) \leftarrow (1-\alpha)\hat{v}_\pi(X_t) + \alpha G_t, \qquad \alpha = \frac{1}{N(X_t)}.$$

☞ In case of non stationary processes, we might consider the general case where $0 < \alpha \leq 1$, i.e., rather consider a running mean.

Robbins Monro conditions

- In the previous updating scheme of the incremental mean, i.e.,

$$\hat{v}_\pi(X_t) \leftarrow (1-\alpha_t)\hat{v}_\pi(X_t) + \alpha_t G_t, \qquad 0 < \alpha_t \leq 1,$$

  and convergence to the true value v_π is not guarantee.
- Sufficient conditions for this, known as Robbins Monro conditions, are

$$\sum_{t\geq 0} \alpha_t = \infty, \qquad \sum_{t\geq 0} \alpha_t^2 < \infty$$

☞ However in concrete implementations we often dot not satisfy such conditions but set $\alpha_t \equiv \alpha$.

Temporal difference learning (TD learning)

- As opposed to Monte Carlo reinforcement learning, TD learning does not require the knowledge of transition matrix and reward function.
- Monte Carlo makes uses of complete episodes, i.e., the sample path of a Markov decision process has reached the terminal state.
- TD learning is able to learn from incomplete episodes

$$v_\pi(x) = \mathbb{E}_\pi(R_{t+1} + \gamma v_\pi(X_{t+1}) \mid X_t = x). \qquad \texttt{(Bellman expectation)}$$

- A temporal difference learning estimator is given by the updating scheme

$$\hat{v}_\pi(X_t) \leftarrow (1-\alpha)\hat{v}_\pi(X_t) + \alpha \left\{R_{t+1} + \gamma \hat{v}_\pi(X_{t+1})\right\}.$$

- Temporal difference learning updates v_π using the estimated return

$$R_{t+1} + \gamma \hat{v}_\pi(X_{t+1}) \approx \mathbb{E}_\pi(R_{t+1} + \gamma v_\pi(X_{t+1}) \mid X_t = x),$$

while Monte Carlo updates v_π from the current return

$$G_t \approx \mathbb{E}_\pi(G_t \mid X_t = x).$$

Algorithm 4: Every visit Monte Carlo prediction of v.

input : The policy π to be evaluated, an initial value function v and a step size $\alpha \in (0, 1]$.

output: An estimate v of v_π.

for *each episode* **do**
  $G \leftarrow 0$;
  $\tilde{\gamma} \leftarrow 1$;
  for *each step of episode* **do**
    $G \leftarrow G + \tilde{\gamma} R$; `// R is current reward`
    $\tilde{\gamma} \leftarrow \gamma \times \tilde{\gamma}$;
    $v(X) \leftarrow (1 - \alpha) v(X) + \alpha G$;
    if X *is a terminal state* **then**
      Go to next episode;
Return the value function v;

Algorithm 5: TD–learning algorithm to estimate v_π.

input : The policy π to be evaluated, an initial value function v and a step size $\alpha \in (0, 1]$.

output: An estimate v of v_π.

for *each episode* **do**
  for *each step of episode* **do**
    Select action $A \leftarrow \pi(X)$; `// X is current state`
    Take action A and observe reward and next state R and X';
    $v(X) \leftarrow (1 - \alpha)v(X) + \alpha \{R + \gamma v(X')\}$;
    $X \leftarrow X'$;
    if X *is a terminal state* **then**
      Go to next episode;
Return the value function v;

- Previous slides were about estimating the value function of an unknown Markov decision process.
- We now jump to the control of Markov decision processes, i.e., estimate the optimal policy of an unknown Markov decision process.
- There are two different situations:

  on–policy where you learn about π from episodes sampled from π;
  off–policy where you learn about π from episodes samples from $\tilde{\pi}$.

q–learning

- Recall that our greedy policy improvement (using $v(x)$) is given by

$$\pi'(x) = \arg\max_{a \in \mathcal{A}} \mathbb{E}_\pi(R_{t+1} + \gamma v(X_{t+1}) \mid X_t = x).$$

- However to compute such expectations we need to know the transition kernel of our Markov decision process $p(\cdot \mid x, a)$.
- Using the action–value function instead we have

$$\pi'(x) = \arg\max_{a \in \mathcal{A}} q(x, a),$$

which is model free in the sense that we do not need to know $p(\cdot \mid x, a)$ but "just" need to estimate $q(\cdot, \cdot)$.

Algorithm 6: The q–learning algorithm.

input : Step size $\alpha \in (0, 1]$, an initial action value function q.
output: An estimate π of π_*.

1 **for** *each episode* **do**
2 **for** *each step of episode* **do**
3 Select action A given X following policy derived from q; `// X is current state`
4 Take action A and observe the reward and next state R and X';
5 $q(X, A) \leftarrow (1 - \alpha)q(X, A) + \alpha \{R + \gamma \max_a q(X', a)\}$;
6 $X \leftarrow X'$;
7 **if** X *is a terminal state* **then**
8 Go to next episode;
9 Derive the optimal policy from q;

The q–learning algorithm is off–policy as a at l.6 doesn't necessarily follow the current policy π.


EXPLORATION
EXPLOITATION

Exploration with an ε–greedy policy

$$q(X,A) \leftarrow (1-\alpha)q(X,A) + \alpha\left\{R + \gamma \max_a q(X',a)\right\}, \qquad A = \arg\max_a q(X,a).$$

Exploration with an ε–greedy policy

$$q(X,A) \leftarrow (1-\alpha)q(X,A) + \alpha\left\{R + \gamma \max_a q(X',a)\right\}, \qquad A = \arg\max_a q(X,a).$$

- Being completely deterministic prevent to explore the whole space $\mathcal{X} \times \mathcal{A}$ required to get accurate estimate of $\max q(X',a)$.
- To mitigate this issue (and to get consistent estimator) we often use a ε–greedy policy improvement where at each state X_t we select action

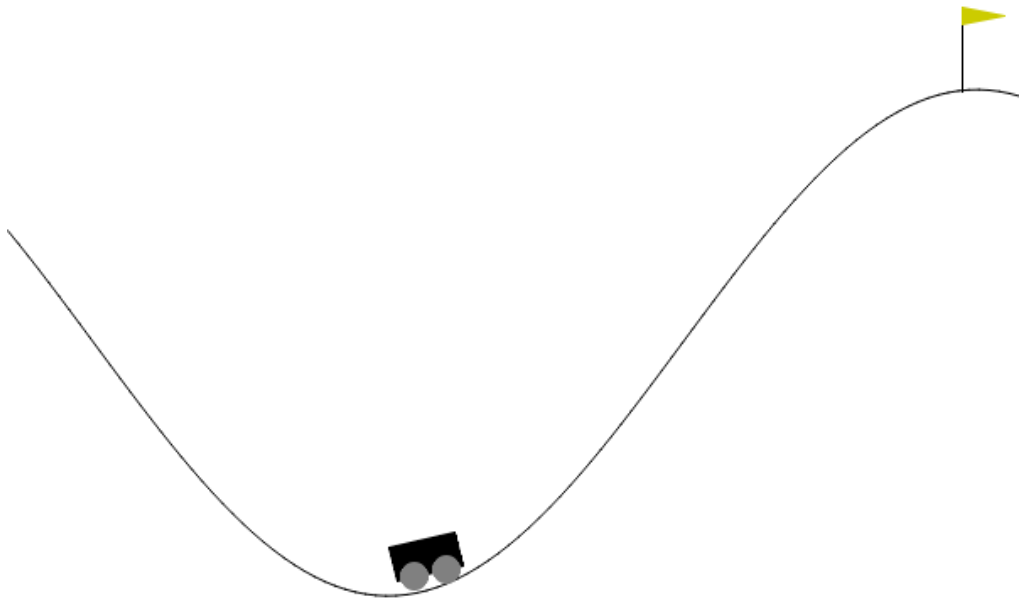
$$A \leftarrow \begin{cases} \text{Discrete}(\mathcal{A}), & \text{with probability } \varepsilon \\ \arg\max_{a \in \mathcal{A}} q(X_t,a), & \text{with probability } 1-\varepsilon \end{cases}$$

Algorithm 7: The q–learning algorithm (ε–greedy).

input : Step size $\alpha \in (0, 1]$, an initial action value function q and an exploration rate $\varepsilon > 0$.
output: An estimate π of π_*.

for *each episode* **do**
  for *each step of episode* **do**
    $U \leftarrow U(0, 1)$;
    if $U < \varepsilon$ **then**
      Select action A at random in $\mathcal{A}$;
    else
      Select action A given X following policy derived from q; `// X is current state`
    Take action A and observe the reward and next state R and X';
    $q(X, A) \leftarrow (1 - \alpha)q(X, A) + \alpha \{R + \gamma \max_a q(X', a)\}$;
    $X \leftarrow X'$;
    if X *is a terminal state* **then**
      Go to next episode;
Derive the optimal policy from q;

Mountain Car problem



- Deterministic Markov Decision Process
- Goal: Reach the flag as quickly as possible
- States: Position along the x–axis and velocity
- Actions: Accelerate to the left, to the right or do not accelerate
- Reward is -1 at each timestep

Mountain Car: Learning stage (Q-learning)

Figure 9: *Click on me to see a movie!*

Mountain Car: Optimal policy

Figure 10: *Click on me again to see another movie!*

☞ Playing 500 games w.r.t. this optimal policy, we won 100% of times! Note how the goal was reached in fewer steps than at episode 250 during the learning stage...

State Action Reward State Action is an alternative to q–learning

Algorithm 8: The SARSA algorithm.

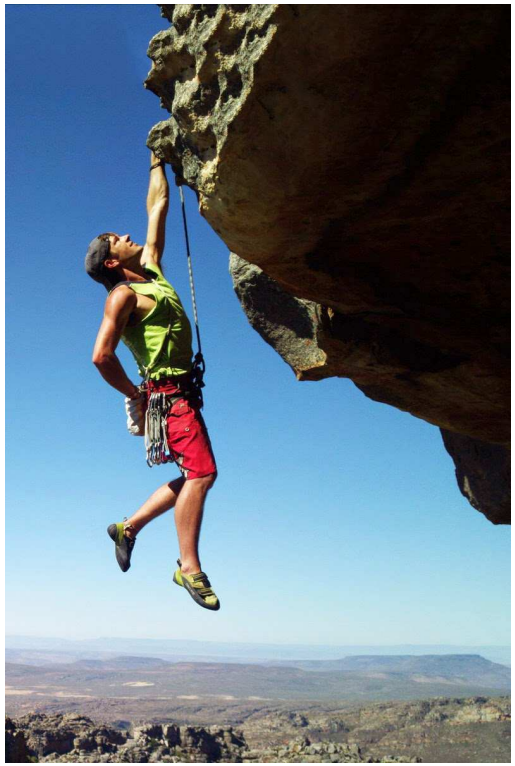
input : Step size $\alpha \in (0, 1]$, an initial action value function q and a tolerance value $\varepsilon > 0$.
output: An estimate π of π_*.

1 **for** *each episode* **do**
2 Select action A given X following policy derived from q (e.g., ϵ–greedy); `// X is initial state`
3 **for** *each step of episode* **do**
4 Take action A and observe the reward and next state R and X';
5 Select action A' given X' following policy derived from q (e.g., ϵ–greedy);
6 $q(X, A) \leftarrow (1 - \alpha)q(X, A) + \alpha \{R + \gamma q(X', A')\}$;
7 $X \leftarrow X'$;
8 $A \leftarrow A'$;
9 **if** X *is a terminal state* **then**
10 Go to next episode;
11 Derive the optimal policy from q;

The SARSA algorithm is on–policy as A' at line I.7 follows the current policy π.

q–learning vs. $SARSA$

- q–learning is off-policy learning while $SARSA$ is on–policy learning.
- With ε–greedy strategies, q–learning estimates the optimal policy while $SARSA$ does not (unless you let $\varepsilon_t \searrow 0$ as $t \to \infty$)
- SARSA is more conservative, i.e., if the "optimal policy is close to a dangerous zone" it will deviate from this optimal policy to less risk adversarial policies (see the following slides about the Cliff example)



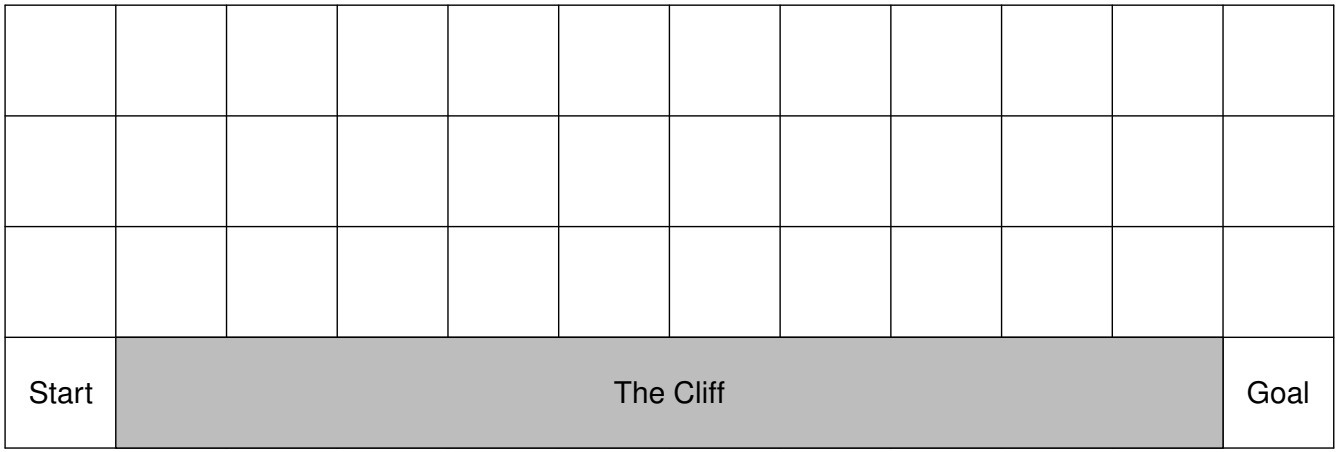


Figure 11: *Cliff walking example. Example 6.6 of Sutton and Barto.*

- Similar to the gridworld example, i.e., $\mathcal{A} = \{north, south, east, west\}$.
- Reward of -1 on all transition except those into the region "The Cliff"
- Stepping into "the Cliff" induces a reward of -100 and sends the agents instantly back to the start.

Figure 12: *Performance of the $SARSA$ and q–learning learning algorithm on the Cliff example. Settings: discount factor $\gamma = 1$, exploration rate $\varepsilon = 0.1$ and step size $\alpha = 0.5$.*

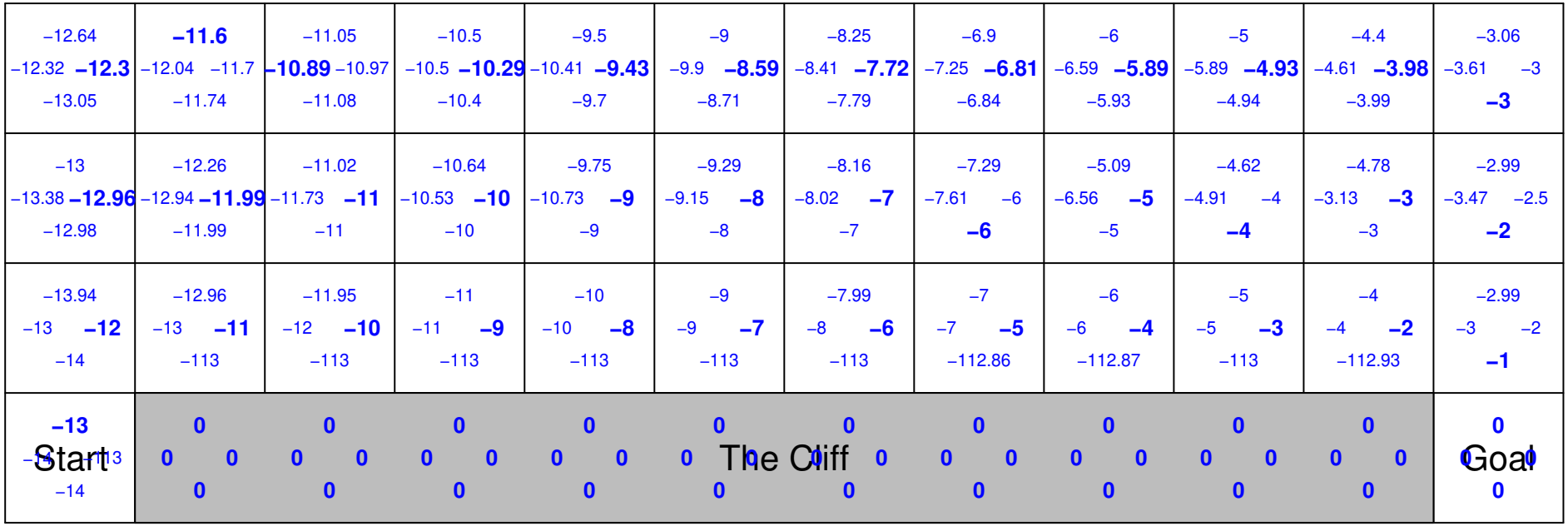


Figure 13: *Estimate of the action–state value function q on the Cliff example using a q–learning algorithm.*

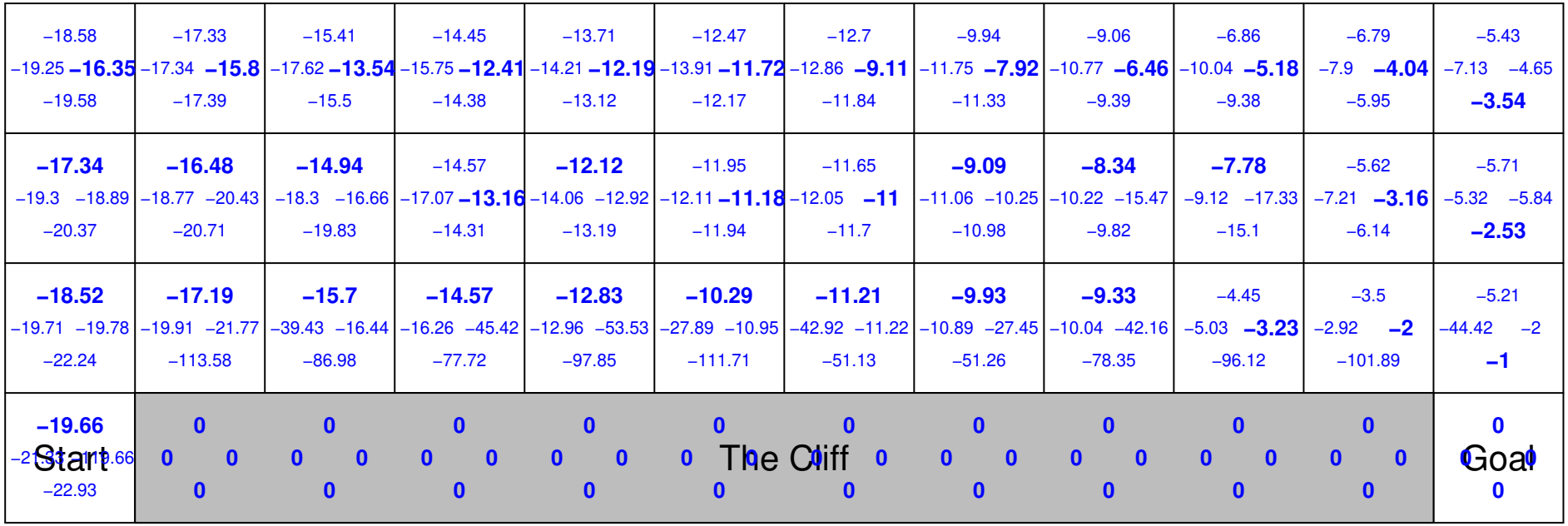


Figure 14: *Estimate of the action–state value function q on the Cliff example using a $SARSA$ algorithm.*

4. Going deep

- We have learned how to estimate the state–action value function
- However both q–learning and $SARSA$ algorithms require to store a lookup table

$$Q = \{q(x, a) \colon (x, a) \in \mathcal{X} \times \mathcal{A}\},$$

and prevent their use when $\dim(\mathcal{X} \times \mathcal{A}) \gg 1$.

- We have learned how to estimate the state–action value function
- However both q–learning and $SARSA$ algorithms require to store a lookup table

$$Q = \{q(x, a) \colon (x, a) \in \mathcal{X} \times \mathcal{A}\},$$

  and prevent their use when $\dim(\mathcal{X} \times \mathcal{A}) \gg 1$.
- Why not trying to estimate the mapping

$$\begin{aligned} q \colon \mathcal{X} \times \mathcal{A} &\longrightarrow \mathbb{R} \\ (x, a) &\longmapsto q(x, a) \end{aligned}$$

  from a parametric statistical model $q(x, a; \theta)$?

A kind of regression problem

- It sounds like a regression problem, i.e.,

$$\arg\min_{\theta\in\Theta} \frac{1}{2}\mathbb{E}\left[\{q(X_t, A_t; \theta) - q_*(X_t, A_t)\}^2\right].$$

- A stochastic gradient descent on θ thus writes

$$\theta^{(k+1)} \leftarrow \theta^{(k)} - \eta\{q(X_t, A_t; \theta^{(k)}) - q_*(X_t, A_t)\}\nabla_\theta q(X_t, A_t; \theta^{(k)}).$$

- However there is a subtlety here since q_* is unknown so that we have to substitute it with an estimate, i.e.,

$$\hat{q}_*(X_t, A_t) = \begin{cases} G_t, & \text{Monte Carlo based estimate} \\ R_{t+1} + \gamma \max_{a'} \hat{q}(X', a'; \theta^{(k)}), & q\text{–learning based estimate} \\ R_{t+1} + \gamma \hat{q}(X_{t+1}, A_{t+1}; \theta^{(k)}), & \text{SARSA based estimate} \end{cases}$$

A "by the way" slide...

- Consider the stupid (but illuminating) situation where

$$q(x, a; \theta) = \theta_{(x,a)}, \qquad x \in \mathcal{X}, a \in \mathcal{A}\},$$

  i.e., as many parameters as q–values.
- We thus have

$$\begin{aligned}\theta^{(k+1)} &= \theta^{(k)} - \eta\{q(X_t, A_t; \theta^{(k)}) - q_*(X_t, A_t)\}\nabla_\theta q(X_t, A_t; \theta^{(k)}) \\ &= \theta^{(k)} - \eta(\theta^{(k)} - \mathbf{Q}_*)^\top e_{(X_t,A_t)}, \qquad e_j \ j\text{-th canonical vector},\end{aligned}$$

  or equivalently, using current state (X_t, A_t),

$$\theta^{(k+1)}_{(X_t,A_t)} \leftarrow (1-\eta)\theta^{(k)}_{(X_t,A_t)} + \eta q_*(X_t, A_t).$$

- Recall we found that

$$\theta^{(k+1)}_{(X_t,A_t)} \leftarrow (1-\eta)\theta^{(k)}_{(X_t,A_t)} + \eta q_*(X_t, A_t).$$

- But remember q_* is unknown but if we substitute it for a q–learning based estimate we get

$$\begin{aligned}\theta^{(k+1)}_{(X_t,A_t)} &\leftarrow (1-\eta)\theta^{(k)}_{(X_t,A_t)} + \eta(R_{t+1} + \gamma \max_{a'} \theta^{(k)}_{(X',a')}) \\ \iff q(X_t, A_t) &\leftarrow (1-\eta)q(X_t, A_t) + \eta(R_{t+1} + \gamma \max_{a'} q(X', a'),\end{aligned}$$

  which is exactly the q–learning updating scheme.
- We thus recover our q–learning algorithm seen as a stochastic gradient descent.

Deep reinforcement learning

- A very hot topic right now is to use (deep) neural networks within a reinforcement learning framework.
- Although it can be used for planning, i.e., estimate the value function for a given policy, we will focus on control only, i.e., get optimal policy.
- To this aim we will try to train a neural network to approximate the action–state value function $q(x, a)$.

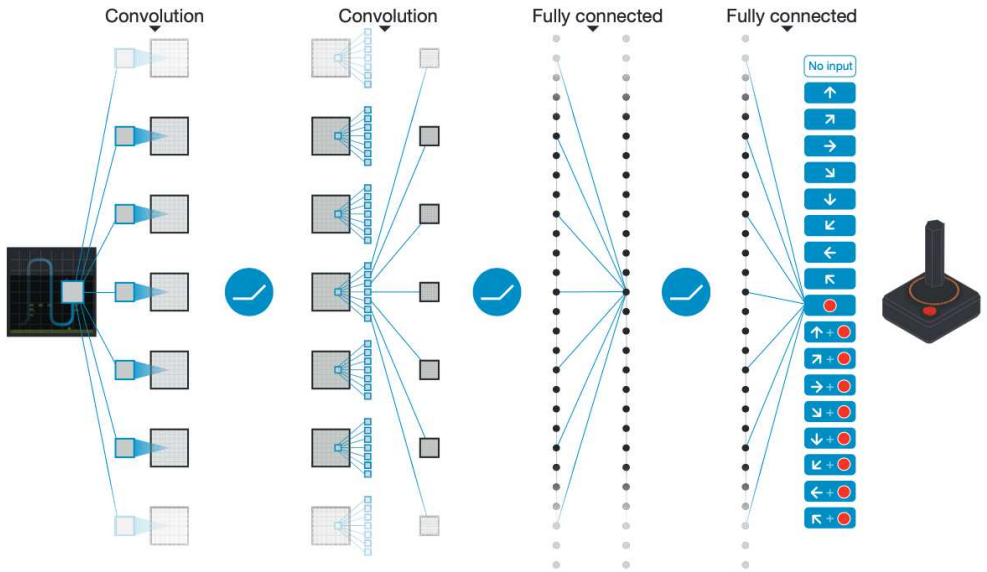


Figure 15: *Illustration of the use of a neural network to estimate the q function for Atari games. [Figure taken from Mnih et al., 2015]*

Algorithm 9: A basic Deep Reinforcement q–learning algorithm.

input : Step size $\alpha \in (0, 1]$, an initial neural net weight θ.
output: An estimate π of π_*.

for *each episode* **do**
  Select action A given X following policy derived from q (e.g., ε–greedy) ; `// X is initial state`
  for *each step of episode* **do**
    Do a forward pass to compute $q(X, A; \theta)$;
    Take action A and observe the reward and next state R and X';
    Do a forward pass to compute $q(X', a')$, $a' \in \mathcal{A}$;
    Perform backpropagation to compute $\nabla_\theta q(X, A; \theta)$;
    Update the neural net parameters
    $\theta \leftarrow \theta - \eta \left\{ q(X, A; \theta) - R - \gamma \max_{a'} q(X', a'; \theta) \right\} \nabla_\theta q(X, A; \theta)$;
    $X \leftarrow X'$;
    if X *is a terminal state* **then**
      Go to next episode;
Derive the optimal policy from q;

- The previous algorithm has weaknesses and is unlikely to converge due to:
  - serial dependence within episode
  - the use of $\hat{q}$ in place of q.
- Current workaround to fix these issues are:
  - Experience replay
  - Fixed q–target

Experience replay

- Simple idea just store past history within a buffer $\mathcal{B} = \{(X_i, A_i, R_{i+1}, X_{i+1}, A_{i+1}) : i = t - B, \ldots, t\}$.
- Sample uniformly, possibly using batches, from the buffer $\mathcal{B}$ to update the gradient in our stochastic gradient descent.
- By doing so you clearly reduce serial dependence.

Fixed q–target

$$\theta \leftarrow \theta - \eta \left\{ q(X, A; \theta) - R - \gamma \max_{a'} q(X', a'; \theta) \right\} \nabla_\theta q(X, A; \theta)$$

- This update may cause troubles since $q(\cdot, \cdot; \theta)$ is used both for:
  - estimating $q(\cdot, \cdot)$;
  - computing the gradient
- Further there is no guarantee that it yields a contraction. . .
- One way to mitigate this side effect is to use two separate networks:

  A target network used for prediction, i.e., the $R + \gamma \max_{a'} q(X', a'; \theta)$ part;
  A q network used for estimation, i.e., the $q(X, A; \theta)$ part;

- Both nets share the same architecture but the target network parameters are updated every C iterations by using that of the q network.

Algorithm 10: Deep Reinforcement q–learning algorithm.

input : Step size $\alpha \in (0, 1]$, an initial neural net weight θ.
output: An estimate π of π_*.

Allocate the size of the replay buffer $\mathcal{B}$;
Initialize the q estimate from the q network using random weights θ;
Initialize the $\tilde{q}$ prediction from the target network with weights $\tilde{\theta} = \theta$;
for *each episode* **do**
    Select action A given X following policy derived from q (e.g., ε–greedy);
    for *each step of episode* **do**
        Take action A and observe the reward and next state R and X';
        Do a forward pass to compute $q(X, A; \theta)$;
        Store sequence $\{(X, A, R, X')\}$ in buffer $\mathcal{B}$;
        Sample random (mini batch of) transitions in $\mathcal{B}$;
        Compute predictions $\{\tilde{q}(X', a; \tilde{\theta}) : a \in \mathcal{A}\}$ using a forward pass from the target network;
        Update the q network weights using a single gradient descent step.
        Every C steps, reset $\tilde{\theta} = \theta$;
        $X \leftarrow X'$;
        if X *is a terminal state* **then**
            Go to next episode;
Derive the optimal policy from q;

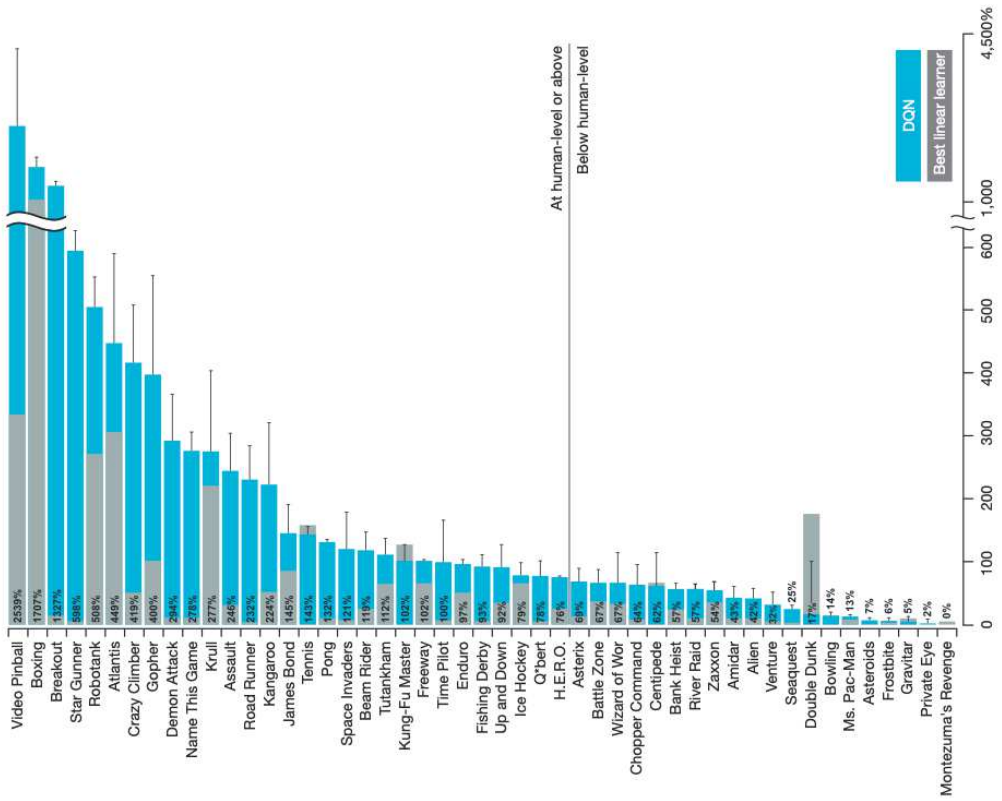
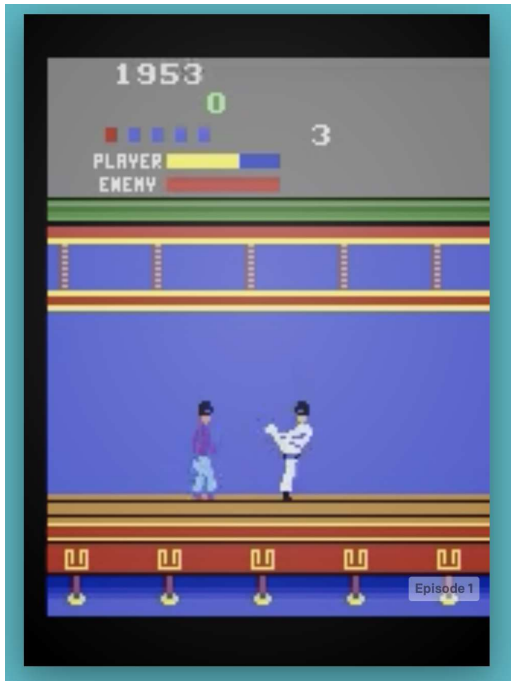


Figure 16: *Performance of Deep q–learning (and linear q–learning) accross various Atari games. Performance is defined as $100 \times (DQN\ score - random\ play\ score)/(Professional\ tester\ score - random\ play\ score)$. [Figure taken from Mnih et al., 2015]*

Implementation details



- We aim at learning $q(X, a)$ using neural nets
- States are images so use of CNN highly recommended
- Reward is the change in score
- State is a stack of raw pixels from last 4 frames. Why?

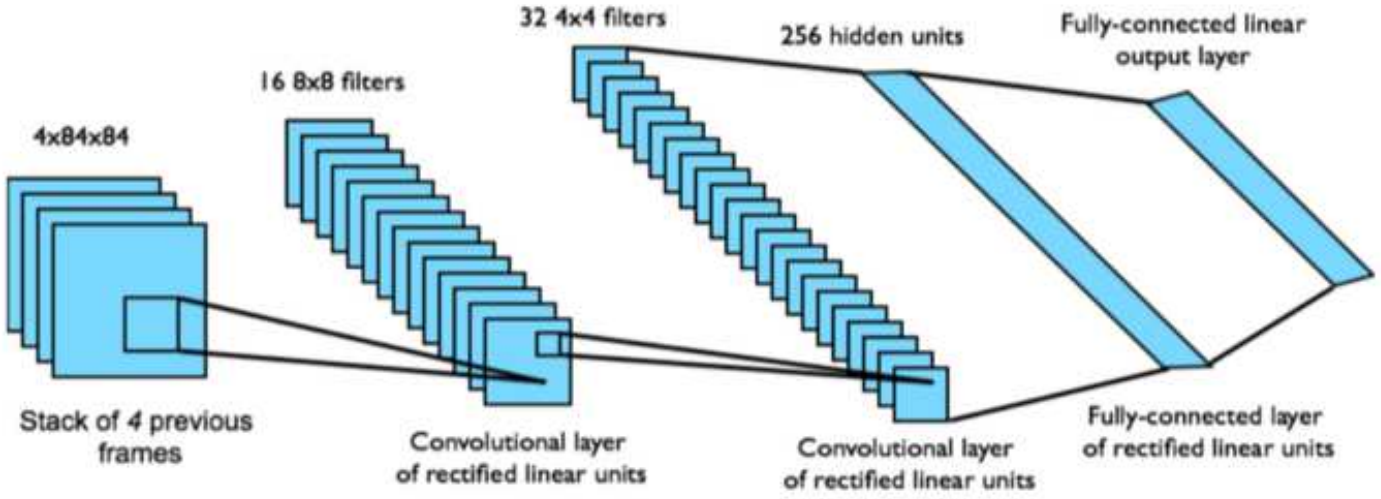


Figure 17: *A typical CNN architecture for the Atari reinforcement learning task. [Picture taken from Emma Brunskill lecture notes]*

5. Policy gradient

Policy gradient methods

- So far we derive the (estimated) optimal policy based on value functions v, q.
- We now rely on parametrized policies

$$\pi_\theta(a \mid x) = \Pr_\pi(a \mid x; \theta), \qquad x \in \mathcal{X}, \quad a \in \mathcal{A},$$

  and retrieve the optimal policy maximizing a performance measure $J(\theta)$.
- For instance one can use gradient ascent, i.e.,

$$\theta_{t+1} = \theta_t + \eta \nabla_\theta J(\theta_t).$$

- Such approaches are called policy gradient methods.

Note that we are working under a model free framework.

Advantages

- Optimizing w.r.t. a parameterized policy may be easier than maximizing value functions
- Can estimate optimal stochastic policies
- May use prior or expert information on the policy through its parametrization.
- Work with finite state / action spaces or infinite ones.

Policy gradient theorem

Theorem 2. *Taking as performance measure $J(\theta) = v_\pi(x_o)$ for some $x_0 \in \mathcal{X}$ and where each episode starts at x_0, we have*

$$\nabla J(\theta) \propto \sum_{x \in \mathcal{X}} \mu(x) \sum_{a \in \mathcal{A}} \nabla \pi(a \mid x) q_\pi(x, a),$$

where π is the policy with parameter θ, μ is the on–policy distribution under π, i.e.,

$$\mu(x) = \frac{\eta(x)}{\sum_{x' \in \mathcal{X}} \eta(x')}, \qquad x \in \mathcal{X},$$

where $\eta(x) = \mathbb{E}(\text{time spent on } x)$.

Proof. If enough time (but start from ∇J and use the relationship between v_π and $q_\pi \ldots$) □

Towards the REINFORCE algorithm

- We just show that

$$\nabla J(\theta) \propto \sum_{x \in \mathcal{X}} \mu(x) \sum_{a \in \mathcal{A}} \nabla \pi(a \mid x) q_\pi(x, a),$$

- The above expression is actually an expection w.r.t. μ, i.e.,

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left[\sum_{a \in \mathcal{A}} \nabla \pi(a \mid X_t) q_\pi(X_t, a) \right], \qquad X_t \sim \mu$$

- It is limited though since we have to marginalize w.r.t. all possible actions.
- Such approaches are called all–action reinforce methods (not used in practice)

Classical REINFORCE

- How to avoid the above marginalization?

Classical REINFORCE

- How to avoid the above marginalization?
- Doing as before but for actions we can write

$$\begin{aligned}\nabla J(\theta) &\propto \mathbb{E}_\pi\left[\sum_{a\in\mathcal{A}} q_\pi(X_t,a)\nabla\pi(a \mid X_t)\right]\\ &\propto \mathbb{E}_\pi\left[\sum_{a\in\mathcal{A}} q_\pi(X_t,a)\frac{\nabla\pi(a \mid X_t)}{\pi(a \mid X_t)}\pi(a \mid X_t)\right]\\ &\propto \mathbb{E}_\pi\left[G_t\frac{\nabla\pi(A_t \mid X_t)}{\pi(X_t,A_t)}\right], \qquad \mathbb{E}_\pi(G_t \mid X_t, A_t) = q(X_t,A_t)\end{aligned}$$

- We can thus use a stochastic gradient ascent scheme, i.e.,

$$\theta_{t+1} = \theta_t + \eta G_t \frac{\nabla\pi(A_t \mid X_t)}{\pi(X_t,A_t)}.$$

A Monte Carlo like learning algorithm

- Recall our updating scheme

$$\theta_{t+1} = \theta_t + \eta G_t \frac{\nabla \pi(A_t \mid X_t)}{\pi(X_t, A_t)}.$$

- Note that it depends on G_t, i.e., the overall reward from time t.
- Hence to be able to get a realization of G_t, the episode should be completed.
- It is thus a Monte Carlo learning algorithm as defined previously in the lecture since we need a complete episode for the updating stage.

Algorithm 11: REINFORCEMENT algorithm

input : A differentiable policy parametrization $\pi_\theta(a \mid x)$, learning rate $\eta > 0$, initial policy parameter θ.

output: Optimal parameter θ

for *each episode* **do**
  Generate a complete episode $X_0, A_0, R_1, \ldots, X_{T-1}, A_{T-1}, R_T$ following $\pi(\cdot \mid \cdot, \theta_t)$;
  for *each step of episode* $t = 0, \ldots, T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
    $\theta \leftarrow \theta + \eta \gamma^t G \nabla \log \pi(A_t \mid X_t, \theta_t)$;
Return the optimal parameter θ and optimal policy $\pi_\theta(\cdot \mid \cdot)$;

Short corridor example (Sutton and Barto)

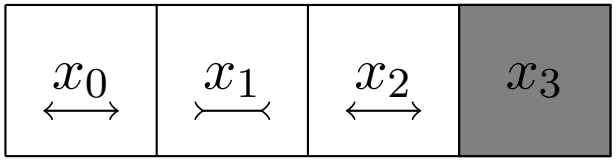


Figure 18: *Small corridor problem.*

- For this problem we have:
  - initial state is x_0 and terminal state is x_3;
  - each move has reward -1.
  - moving left at x_0 causes no movement;
  - moving left at x_1 actually move at right (and vice versa).
- Performance measure is set to $J(\theta) = v_\theta(x_0)$.
- We use the following (redundant) parametrization

$$\pi_\theta(x, \text{right}) \propto \exp(\theta_1), \qquad \pi_\theta(x, \text{left}) \propto \exp(\theta_2),$$

  which is independent of x.

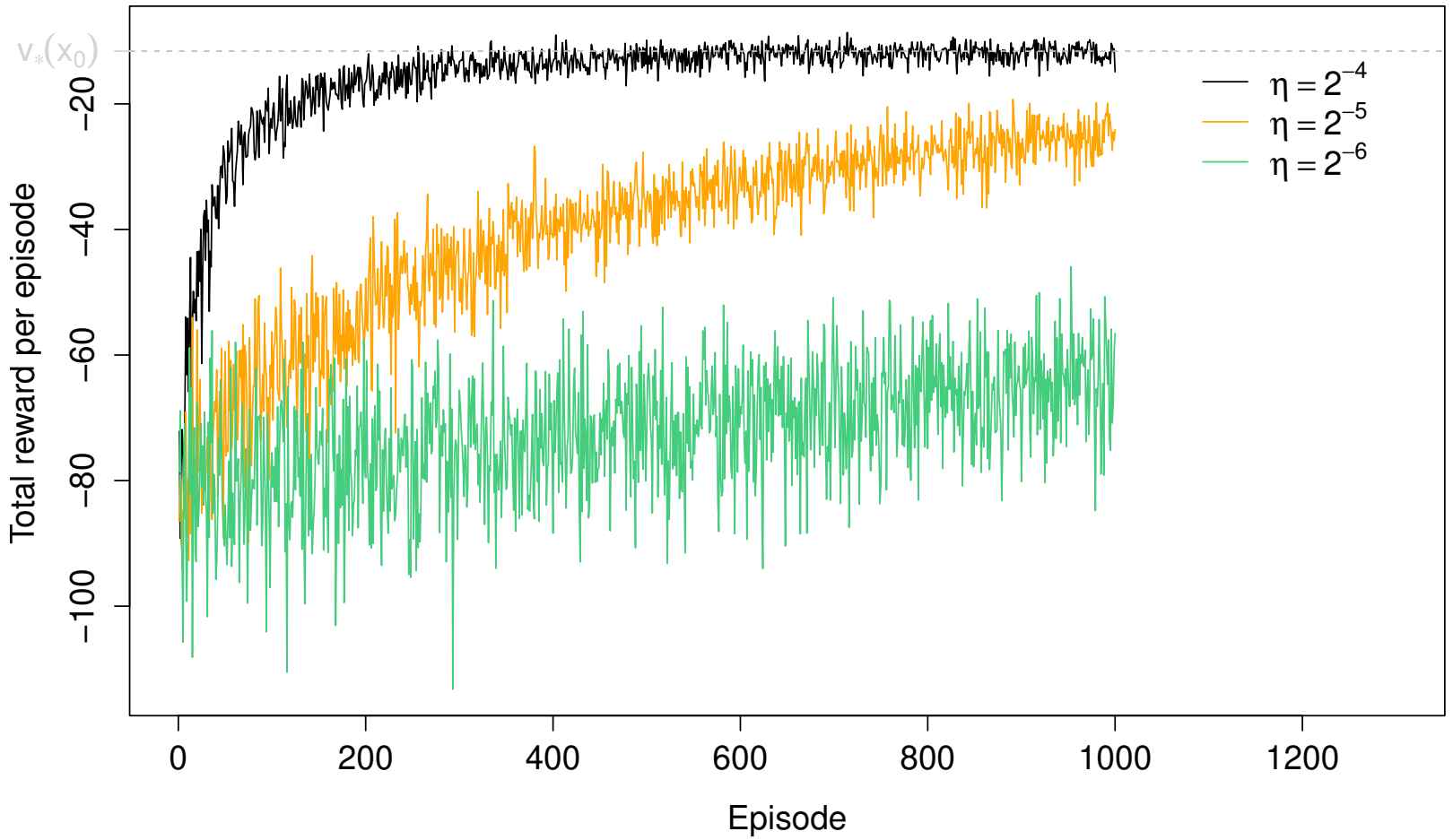


Figure 19: *Learning the optimal stochastic policy using a REINFORCEMENT learning strategy. Shown are the evolution of the performance measure $J(\theta)$ through iterations for different learning rates.*

REINFORCEMENT with Baseline

- We can generalize the REINFORCEMENT algorithm using a baseline, i.e.,

$$\nabla J(\theta) \propto \sum_x \mu(x) \sum_a \{q_\pi(x,a) - b(x)\} \nabla \pi(a \mid x, \theta).$$

- No impact on the learning as long as $b(s)$ is independent of a since

$$\sum_a b(x) \nabla \pi(a \mid x, \theta) = b(x) \sum_a \nabla \sum_a \pi(a \mid x, \theta) = b(s) \nabla 1 = 0.$$

- The updating stage in the gradient ascent is now

$$\theta_{t+1} = \theta_t + \eta \{G_t - b(X_t)\} \nabla \log \pi(A_t \mid X_t, \theta)$$

- A typical choice is a parameterized state value function $b(x) = \hat{v}(x, \psi)$.

☞ Using a baseline can significantly speed up the learning stage!

Algorithm 12: REINFORCEMENT algorithm with baseline $\hat{v}(x, \psi)$

input : A differentiable policy and satet value function parametrizations $\pi_\theta(a \mid x)$, $\hat{v}(x, \psi)$, learning rates $\eta_\theta > 0$ and $\eta_\psi > O$, initial policy and state value parameters θ, ψ.

output: Optimal parameter θ

for *each episode* **do**
  Generate a complete episode $\{(X_t, A_t, R_{t+1}) \colon t = 1, \ldots, \text{end epidose}\}$;
  for *each step of episode* $t = 0, \ldots, T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
    $\delta \leftarrow G - \hat{v}(X_t, \psi)$;
    `/* Gradient ascent for` $v(\cdot; \psi)$ `*/`
    $\omega \leftarrow \omega + \eta_\psi \delta \nabla \hat{v}(X_t, \psi)$;
    `/* Gradient ascent for` π_θ `*/`
    $\theta \leftarrow \theta + \eta_\theta \gamma^t \delta \nabla \log \pi(A_t \mid X_t, \theta_t)$;
Return the optimal parameter θ and optimal policy $\pi_\theta(\cdot \mid \cdot)$;

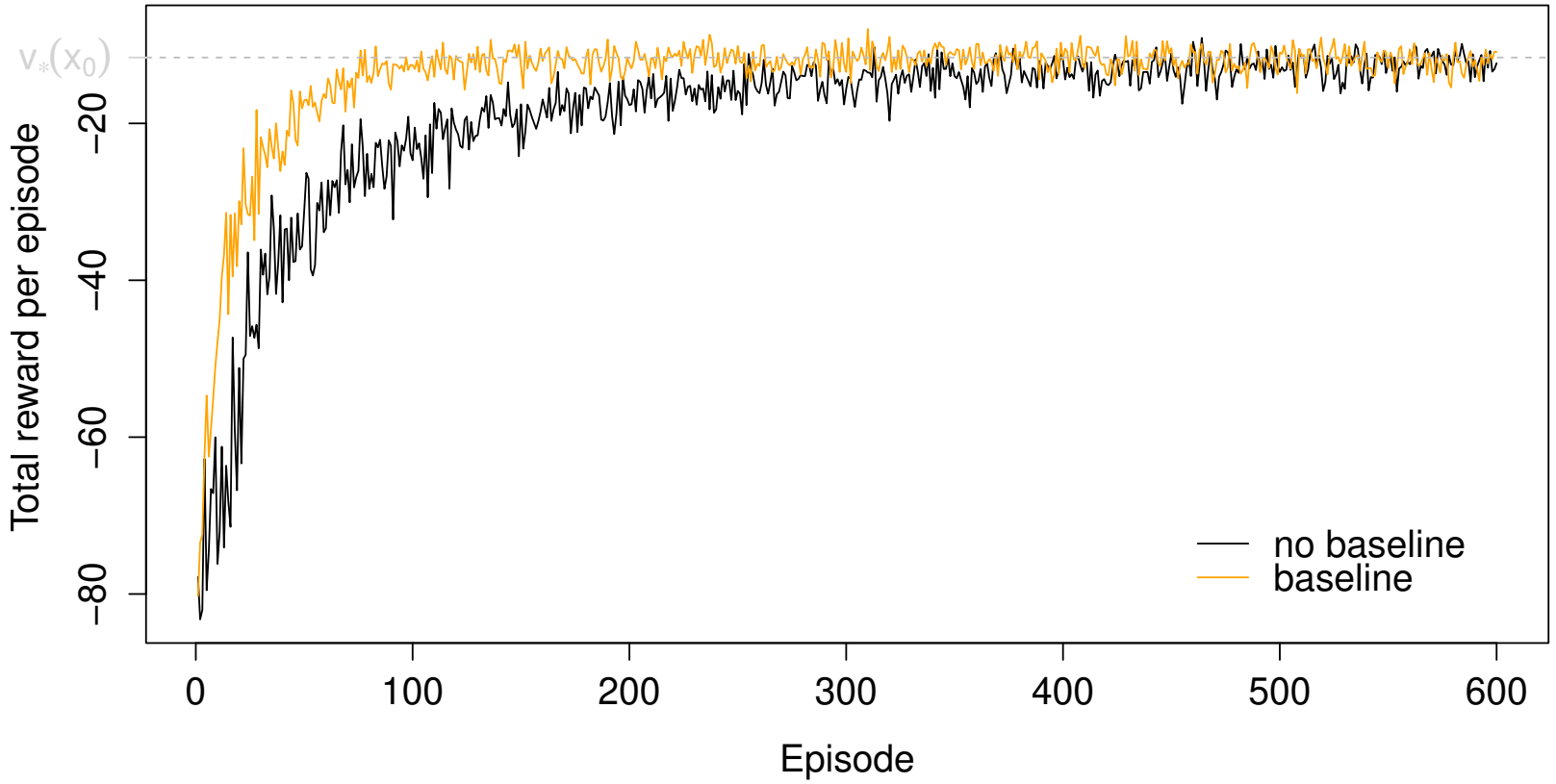


Figure 20: *Learning the optimal stochastic policy using a REINFORCEMENT baseline learning strategy. Shown are the evolution of the performance measure $J(\theta)$ through iterations with baseline and no baseline.*

What we didn't cover

- k–armed bandit
- Double Q–learning
- n–steps TD methods
- Eligibility traces (general model that embeds TD and Monte Carlo learning)
- Actor–Critic methods (similar to REINFORCE)
- For concrete applications:
  - State aggregation (only during Labs)
  - Tile coding (discretization of $\mathcal{X}$)

THIS IS THE END...