

Markov Decision Processes

Johanni Brea

25 April 2023

Artificial Neural Networks CS-456

Introduction

Many RL papers contain a background section like the following one:

The Option-Critic Architecture

Pierre-Luc Bacon, Jean Harb, Doina Precup

Reasoning and Learning Lab, School of Computer Science
McGill University
{pbacon, jharb, dprecup}@cs.mcgill.ca

Preliminaries and Notation

A Markov Decision Process consists of a set of states \mathcal{S} , a set of actions \mathcal{A} , a transition function $P : \mathcal{S} \times \mathcal{A} \rightarrow (\mathcal{S} \rightarrow [0, 1])$ and a reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. For convenience, we develop our ideas assuming discrete state and action sets. However, our results extend to continuous spaces using usual measure-theoretic assumptions (some of our empirical results are in continuous tasks). A (Markovian stationary) *policy* is a probability distribution over actions conditioned on states, $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. In discounted problems, the value function of a policy π is defined as the expected return: $V_\pi(s) = \mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s]$ and its action-value function as $Q_\pi(s, a) = \mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a]$, where $\gamma \in [0, 1)$ is the *discount factor*. A policy π is *greedy* with respect to a given action-value function Q if

In this lecture you will learn

1. what a Markov Decision Process is.
2. how MDPs can be solved with dynamic programming or linear programming.
3. how future discounted MDPs can be solved with value iteration or policy iteration.

Recommended reading:

Sutton & Barto, Chapters 3 & 4

Algorithms of Reinforcement Learning

<http://www.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf>

Markov Decision Processes

We consider **Markov Decision Processes (MDPs)** with

- ▶ finite state space \mathcal{S} with $|\mathcal{S}| < \infty$,
- ▶ finite action spaces $\{\mathcal{A}_s | s \in \mathcal{S}\}$ with $|\mathcal{A}_s| < \infty$,
- ▶ immediate rewards $r_s^a \in \mathbb{R}$
- ▶ transition probabilities $p_{s_i \rightarrow s_j}^a \in [0, 1]$
- ▶ discount factor $\gamma \in [0, 1]$
- ▶ and initial state probabilities $p_{s_i}^{(0)}$.

For a sequence (or trajectory) of state-action-reward tuples, we will use the notation

$$\tau = (S_0, A_0, R_1, S_1, A_1, \dots, R_T).$$

- finite state space \mathcal{S} with $|\mathcal{S}| < \infty$,
- finite action spaces $\{\mathcal{A}_s | s \in \mathcal{S}\}$ with $|\mathcal{A}_s| < \infty$,
- immediate rewards $r_t^a \in \mathbb{R}$,
- transition probabilities $p_{s' \rightarrow s}^a \in [0, 1]$
- discount factor $\gamma \in [0, 1]$
- and initial state probabilities $p_s^{(0)}$.

For a sequence (or trajectory) of state-action-reward tuples, we will use the notation $\tau = \{S_0, A_0, R_0, S_1, A_1, \dots, R_T\}$.

Notes

- The Markov Decision Processes can also be defined for continuous state and action spaces, but we restrict ourselves here to finite (and thus discrete) state and action spaces.
- In general, the available actions can depend on the state (it is not possible to advance when standing in front of a wall). Sometimes the action spaces are independent of the state; in this case we just write \mathcal{A} for the action space.
- The transition probabilities have the property $\sum_{s_j \in \mathcal{S}} p_{s_i \rightarrow s_j}^a = 1, \forall a \in \mathcal{A}_{s_i}, s_i \in \mathcal{S}$.
- Sometimes rewards are considered stochastic or dependent on the next state $R_{s_i \rightarrow s_j}^a$. In this case one can define the immediate rewards as the expected immediate rewards $r_{s_i}^a = E[R_{s_i \rightarrow s_j}^a]$.
- The initial state probabilities have the property $\sum_{s_i \in \mathcal{S}} p_{s_i}^{(0)} = 1$.
- Different authors use different conventions to define MDPs; some include only the state space, action space, transition probabilities and rewards, others include also the discount factor or the initial state probabilities.

Policies, Value Functions and Objectives

The goal is to find a **policy** $\pi^{(t)}(a|s) \in [0, 1]$ (i.e. probability of taking action a in state s and time point t) that maximizes some objective. We use the notation π to denote the policy for all states and time points. We define the horizon- T **value function**

$$\begin{aligned} V_{\gamma}^{(T)}(\pi, s) &= \mathbb{E} \left[\sum_{t=1}^T \gamma^{(t-1)} R_t \mid S_0 = s \right] \\ &= \sum_{A_0, S_1, A_1, \dots, A_{T-1}} \pi^{(0)}(A_0|s) p_{s \rightarrow S_1}^{A_0} \cdots \left(r_s^{A_0} + \gamma r_{S_1}^{A_1} + \cdots + \gamma^{T-1} r_{S_{T-1}}^{A_{T-1}} \right) \end{aligned} \quad (1)$$

Objectives find the policy π that maximizes for all $s \in \mathcal{S}$

- ▶ **Horizon-T values:** $V_{\gamma}^{(T)}(\pi, s)$.
- ▶ **Future Discounted Values:** $V_{\gamma}^{\infty}(\pi, s) = \lim_{T \rightarrow \infty} V_{\gamma}^{(T)}(\pi, s)$ for $\gamma \in [0, 1)$.
- ▶ **Reward Rate:** $\lim_{T \rightarrow \infty} \frac{1}{T} V_1^{(T)}(\pi, s)$.

Notes

The goal is to find a policy $\pi^{(t)}(a|s) \in [0, 1]$ (i.e. probability of taking action a in state s and time point t) that maximizes some objective. We use the notation π to describe the policy for all states and time points. We define the horizon- T value function

$$V_t^{(T)}(s, \pi) = E \left[\sum_{t'=t}^T \gamma^{t'-t} R_{t'} \mid S_t = s \right] \quad (1)$$

$$= \sum_{A_t, A_{t+1}, \dots, A_{T-1}} \gamma^{T-t} P_{A_t, A_{t+1}, \dots, A_{T-1}}(s) \left(R_t + \gamma V_{t+1}^{(T)} + \dots + \gamma^{T-t} R_{T-1} \right)$$

Objectives find the policy π that maximizes for all $s \in \mathcal{S}$

- **Horizon- T values:** $V_t^{(T)}(s, \pi)$.
- **Future Discounted Values:** $V_t^\gamma(s, \pi) = \lim_{T \rightarrow \infty} V_t^{(T)}(s, \pi)$ for $\gamma \in [0, 1]$.
- **Reward Rate:** $\lim_{T \rightarrow \infty} \frac{1}{T} V_t^{(T)}(s, \pi)$.

- In general, the policy can depend on the time point, but in some cases it is independent of time and we can drop the upper index (t).
- On the second line of the definition of the value function we write out explicitly the expectation by summing over all possible actions and states (up to horizon T), weighted by the probabilities of taking those actions $\pi^{(t)}(A_t, S_t)$ (given by the policy) and the transition probabilities $p_{S_t \rightarrow S_{t+1}}^{A_t}$.

Comments

- ▶ Sometimes it will be useful to work with Q-values

$$Q_{\gamma}^{(T)}(\pi, s, a) = \mathbb{E} \left[\sum_{t=1}^T \gamma^{(t-1)} R_t \mid S_0 = s, A_0 = a \right]$$

$$V_{\gamma}^{(T)}(\pi, s) = \sum_{a \in \mathcal{A}_s} \pi(a|s) Q_{\gamma}^{(T)}(\pi, s, a)$$

- ▶ **Why is it called “Markov Decision Process”?** A Markov Decision Process together with a policy defines a Markov chain on space \mathcal{S} with transition probabilities $T_{s_i \rightarrow s_j} = p_{s_i \rightarrow s_j}^a \pi(a|s_i)$.
- ▶ **Are there non-Markovian Decision Processes?** Yes! Depending on how the state space is defined, the next state s_j may depend on more than just the current state and action. For example, in partially observable Markov Decision Processes (POMDPs) one assumes there is an underlying MDP, but instead of observing the full state the agent observes only parts of the full state.

Notes

- Sometimes it will be useful to work with Q-values

$$Q^{\pi}(s, a) = E \left[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = s, A_0 = a \right]$$

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a) Q^{\pi}(s, a)$$

- **Why is it called "Markov Decision Process"?** A Markov Decision Process together with a policy defines a Markov chain on space \mathcal{X} with transition probabilities $T_{s' = \pi(a)}(s, a)$.
- **Are there non-Markovian Decision Processes?** Yes! Depending on how the state space is defined, the next state s_t may depend on more than just the current state and action. For example, in partially observable Markov Decision Processes (POMDP) one assumes there is an underlying MDP but instead of observing the full state the agent observes only parts of the full state.

In the Atari games we encountered already a non-Markovian Decision Process: for moving objects a single video frame is not sufficient to determine the direction of movement. In this case it is easy to turn the non-Markovian Decision Process into a Markov Decision Process by augmenting the state space and defining the state as a few subsequent video frames.

A famous, academic example of a partially observable Markov Decision Process is the so-called Tiger Problem (<https://people.csail.mit.edu/lpk/papers/aij98-pomdp.pdf>), where a tiger is behind one door and a large reward is behind the other door. The agent can either listen, or open the left or the right door. When the agent listens, it observes a roar either behind the left or the right door, but the observation is not always accurate; with a small probability the agent may observe a roar behind the left door, even when the tiger is behind the right door and vice versa.

All non-Markovian Decision Processes could in principle be turned into Markov Decision Processes by augmenting the state space (for example with perfect knowledge about the actual position of the tiger) but in practice it may be difficult or impossible to do this augmentation.

Our world is usually partially observable: as long as the door of the fridge is closed we do not directly observe the content of the fridge; the full state of mind of another person is usually unobservable to us.

What is the relationship to Reinforcement Learning?

“Solving” an MDP amounts to solving an optimal control problem, i.e. finding the optimal policy, where the dynamics is known, i.e. $p_{s_i \rightarrow s_j}^a$ and r_s^a are assumed to be known. In reinforcement learning, one assumes that the dynamics is unknown.

- ▶ In model-free RL, the agent tries to find the optimal policy, without ever explicitly estimating the dynamics $p_{s_i \rightarrow s_j}^a$ and rewards r_s^a .
- ▶ In model-based RL, the agent tries to estimate the dynamics and then solves the control problem.

Notes

"Solving" an MDP amounts to a solving an optimal control problem, i.e. finding the optimal policy, where the dynamics is known, i.e. p_{t+1}^s and r_t^s are assumed to be known. In reinforcement learning, one assumes that the dynamics is unknown.

- In model-free RL, the agent tries to find the optimal policy, without ever explicitly estimating the dynamics p_{t+1}^s and rewards r_t^s .
- In model-based RL, the agent tries to estimate the dynamics and then solves the control problem.

- There is no estimation problem involved in solving an MDP, but solving the optimal control problem is still a non-trivial problem itself.
- Model-based reinforcement learning solves explicitly an estimation problem *and* an optimal control problem. Model-free reinforcement learning solves the estimation and the optimal control problem implicitly.
- The exploration-exploitation trade-off exists only in reinforcement learning, but not when solving MDPs. One could say, solving MDPs is solving an exploitation problem. The exploration part in reinforcement learning is needed to tackle the estimation problem.

The Optimal Fixed Horizon Policy

The policy π^* that maximizes the horizon-T values can be found with **Dynamic Programming**: recursively find the optimum for problems of growing horizon.

1. The optimal horizon-1 values are $V_\gamma^{(1)}(\pi^*, s) = \max_{a \in \mathcal{A}_s} r_s^a$.
2. The optimal horizon- $(t + 1)$ values are

$$V_\gamma^{(t+1)}(\pi^*, s) = \max_{a \in \mathcal{A}_s} Q_\gamma^{(t+1)}(\pi^*, s, a) = \max_{a \in \mathcal{A}_s} r_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{s \rightarrow s'}^a V_\gamma^{(t)}(\pi^*, s') \quad (2)$$

The optimal horizon-T policy picks at time t an action in the

$$\arg \max_{a \in \mathcal{A}_s} Q_\gamma^{(T-t+1)}(\pi^*, s, a).$$

The horizon-T policy is not stationary, in general, i.e. $\pi^{(t)}(a|s) \neq \pi^{(t')}(a|s)$ for $t \neq t'$, but it can be chosen to be deterministic.

Notes

The policy π^* that maximizes the horizon- T values can be found with **Dynamic Programming**: recursively find the optimum for problems of growing horizon.

1. The optimal horizon-1 values are $V_0^*(r^*, a) = \max_{a \in \mathcal{A}_s} r_s^a$.
2. The optimal horizon- $t+1$ values are

$$V_t^{(t+1)}(r^*, a) = \max_{a' \in \mathcal{A}_s} Q_t^{(t+1)}(r^*, a, a') = \max_{a' \in \mathcal{A}_s} r_s^{a'} + \sum_{s'} p_{ss'}^{a'} V_{t-1}^{(t+1)}(r^*, a')$$

The optimal horizon- T policy picks at time t an action in the $\arg \max_{a \in \mathcal{A}_s} Q_t^{(T-t+1)}(r^*, a, a)$.

The horizon- T policy is not stationary, in general, i.e. $\pi^{(t)}(a|s) \neq \pi^{(t')}(a|s)$ for $t \neq t'$, but it can be chosen to be deterministic.

- The **dynamic programming** method breaks decision problems into smaller subproblems. **Bellman's principle of optimality** describes how to do this: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (See Bellman, 1957, Chap. III.3.)
- The $\arg \max_{a \in \mathcal{A}_s} Q_\gamma^{(T-t+1)}(\pi^*, s, a)$ may contain multiple actions. In this case one can randomly break ties and select any action that maximizes the Q-values.
- For a horizon- T problem, one finds first the (set of) optimal action(s) for the last time step (the solution for the horizon-1 problem), for example $\pi^{(T)}(a|s) = 1$ if $a = \text{first}(\arg \max_{a \in \mathcal{A}_s} r_s^a)$ and $\pi^{(T)}(a'|s) = 0$ for all $a' \neq a$.
- Then one finds the (set of) optimal action(s) for the second to last time step (the solution for the horizon-2 problem), e.g. $\pi^{(T-1)}(a|s) = 1$ if $a = \text{first}(\arg \max_{a \in \mathcal{A}_s} Q^{(2)}(\pi^*, s, a))$ and $\pi^{(T-1)}(a'|s) = 0$ for all $a' \neq a$, etc.
- In the exercises you will construct an example to show that the horizon- T policy can be non-stationary.

Fixed-Point Iterations and Banach's Fixed Point Theorem

Some equations of the form $x = T(x)$ can be solved with a fixed point iteration:

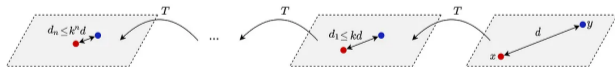
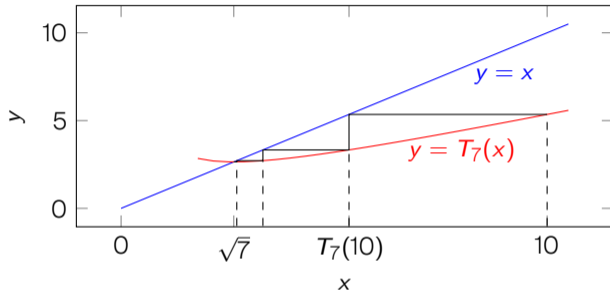
Start with $x^{(0)}$ and compute

$$x^{(k)} = T(x^{(k-1)})$$

until $x^{(k)} \approx x^{(k-1)}$.

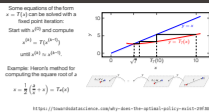
Example: Heron's method for computing the square root of a

$$x = \frac{1}{2} \left(\frac{a}{x} + x \right) = T_a(x)$$



<https://towardsdatascience.com/why-does-the-optimal-policy-exist-29f30fd51f8c>

Notes



For the interested students, here is the mathematical background (see also Appendix A of <http://www.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf>):

Let (X, d) be a complete metric space. Then a map $T : X \rightarrow X$ is called a **contraction mapping** on X if there exists $q \in [0, 1)$ such that $d(T(x), T(y)) \leq qd(x, y)$ for all $x, y \in X$.

Banach Fixed Point Theorem. Let (X, d) be a non-empty complete metric space with a contraction mapping $T : X \rightarrow X$. Then T admits a unique fixed-point x^* in X (i.e. $T(x^*) = x^*$). Furthermore, x^* can be found as follows: start with an arbitrary element $x_0 \in X$ and define a sequence $(x_n)_{n \in \mathbb{N}}$ by $x_n = T(x_{n-1})$ for $n \geq 1$. Then $\lim_{n \rightarrow \infty} x_n = x^*$.

For Heron's method for computing the square root, we can take the distance $d(x, y) = |x - y|$ and the contraction mapping $T : [\sqrt{a/2}, \infty) \rightarrow [\sqrt{a/2}, \infty)$, $x \rightarrow T_a(x)$ for $a > 0$. It is easy to show that $\left| \frac{df_a}{dx}(x) \right| \leq \frac{1}{2}$ on $[\sqrt{a/2}, \infty)$ and therefore (by the mean value theorem) $|f_a(x) - f_a(y)| \leq \frac{1}{2}|x - y|$.

Maximizing Future Discounted Values with Dynamic Programming

Let us define the mapping (sometimes called **Bellman operator**)

$$T_\gamma : \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}, T_\gamma(X)_s = \max_{a \in \mathcal{A}_s} \left(r_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{s \rightarrow s'}^a X_{s'} \right). \quad (3)$$

- ▶ One can show that the mapping T_γ is a contraction mapping and Banach's fixed point theorem can be applied. Hence, there is a unique fixed point $X^* = T_\gamma(X^*)$.
- ▶ Note that $X^* = T_\gamma(X^*)$ is exactly the same equation we want the optimal horizon- ∞ values to satisfy (c.f. Eq. 2).
- ▶ Therefore, this fixed point is the solution $V_\gamma^\infty(\pi^*, s) = X_s^*$.
- ▶ The optimal policy is to choose actions in $\arg \max_{a \in \mathcal{A}_s} Q_\gamma^\infty(\pi^*, s, a)$.
- ▶ This policy is stationary, i.e. $\pi^{(t)}(a|s) = \pi^{(t')}(a|s)$ for $t \neq t'$, and it can be chosen to be deterministic.

Value Iteration

Iteratively compute horizon- t values until $\max_{s \in \mathcal{S}} |V_{\gamma}^{(t+1)}(\pi^*, s) - V_{\gamma}^{(t)}(\pi^*, s)| < \theta$, where $\theta > 0$ is some convergence criterion. The optimal stationary policy picks actions in $\arg \max_{a \in \mathcal{A}_s} Q_{\gamma}^{t^*}(\pi^*, s, a)$, where t^* is the stopping iteration.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

```
|  $\Delta \leftarrow 0$   
| Loop for each  $s \in \mathcal{S}$ :  
|    $v \leftarrow V(s)$   
|    $V(s) \leftarrow \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$   
|    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$   
until  $\Delta < \theta$ 
```

Output a deterministic policy, $\pi \approx \pi_*$, such that
$$\pi(s) = \arg \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$$

Notes

It is not unreasonable to start the fixed-point iteration with $X_S = \max_{a \in \mathcal{A}_s} r_s^a$ (as for the horizon- T solution), but value iteration would also converge, if one initialized X randomly (thanks to the contraction mapping and Banach's fixed point theorem).

Having to define some stopping criterion makes value iteration a bit unattractive.

Value Iteration

Iteratively compute horizon- ∞ value until $\max_{a \in \mathcal{A}_s} |V^{(t+1)}(s^i, a) - V^{(t)}(s^i, a)| < \epsilon$, where $\epsilon > 0$ is some convergence criterion. The optimal stationary policy picks actions in $\arg \max_{a \in \mathcal{A}_s} Q^*(s^i, a)$, where t^* is the stopping iteration.

Value Iteration, for outlining $\epsilon > 0$:

```
Algorithm parameter: a small threshold  $\epsilon > 0$  denoting accuracy of solution
Initialize  $V^0(s)$  for all  $s \in \mathcal{S}$ , arbitrarily except that  $V^0(s_{\text{terminal}}) = R$ 
Loop:
   $\Delta \leftarrow 0$ 
  Enumerate each  $s \in \mathcal{S}$ :
     $v \leftarrow V^0(s)$ 
     $F(s) \leftarrow \max_a \sum_{s'} P(s'|s, a) (R + \gamma V^0(s'))$ 
     $\Delta \leftarrow \max(\Delta, |v - F(s)|)$ 
  until  $\Delta < \epsilon$ 
Output a stationary policy,  $\pi = \pi_t$ , such that
 $v(s) = \max_{a \in \mathcal{A}_s} \sum_{s'} P(s'|s, a) (R + \gamma V^0(s'))$ 
```

Policy Iteration

Policy Improvement Theorem

Let π and π' be a pair of deterministic policies such that $\forall s$

$$Q_{\pi}(s, \pi'(s)) \geq Q_{\pi}(s, \pi(s)).$$

Then the policy π' must be as good as or better than π , i.e.

$$Q_{\pi'}(s, \pi'(s)) \geq Q_{\pi}(s, \pi(s)).$$

Policy Evaluation

$$V_{\gamma}^{\infty}(\pi, s) = r_s^{\pi(s)} + \gamma \sum_{s' \in \mathcal{S}} p_{s \rightarrow s'}^{\pi(s)} V_{\gamma}^{\infty}(\pi, s') \quad (4)$$

$$\mathbf{V}_{\gamma}^{\infty}(\pi) = (I - \gamma P)^{-1} \mathbf{r} \quad (5)$$

with $\mathbf{r}_s = r_s^{\pi(s)}$ and identity matrix I and $P_{s,s'} = p_{s \rightarrow s'}^{\pi(s)}$.

Policy Improvement

$$\pi'(s) = \text{first}(\arg \max_{a \in \mathcal{A}_s} Q_{\gamma}^{\infty}(\pi, s, a))$$

Policy Iteration

Start with a random deterministic policy, evaluate it, improve it and repeat evaluation and improvement until the policy does not change anymore.

Notes

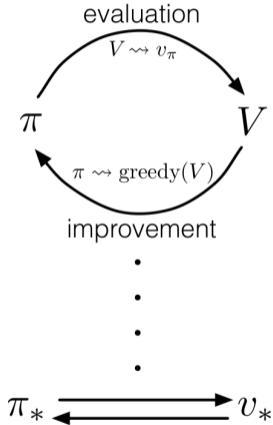
- For deterministic policies $\pi(a|s) = 1$ for some a and $\pi(a'|s) = 0, \forall a' \neq a$, we will use here the notation $a = \pi(s)$.
- The policy evaluation can be done by explicitly inverting the matrix $(I - \gamma P)$ or by approximating the Neumann series $(I - \gamma P)^{-1} = \sum_{k=0}^{\infty} \gamma^k P^k$ as in the example of iterative policy evaluation in the pseudocode on the right.

Policy Evaluation
 $V^{\pi}(s, a) = v_s^{\pi} + \sum_{s'} P_{ss'}^{\pi}(a) V^{\pi}(s', a)$ (H)
Let π and π' be a pair of deterministic policies such that $\forall s, Q_{\pi}(s, \pi'(s)) \geq Q_{\pi}(s, \pi(s))$, with $v_s = v_s^{\pi}$ and identify matrix I and $P_{ss'} = P_{ss'}^{\pi}$.
Then the policy π' must be as good as or better than π , i.e. $v^{\pi'}(s) = \text{fixed point}_{v, a} Q_{\pi'}^{\pi}(s, a, a)$.
Policy Iteration
Start with a random deterministic policy, evaluate it, improve it and repeat evaluation and improvement until the policy does not change anymore.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
 $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
2. Policy Evaluation
Loop:
 $\Delta \leftarrow 0$
Loop for each $s \in \mathcal{S}$:
 $v \leftarrow V(s)$
 $V(s) \leftarrow \sum_{s', r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$
 $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
3. Policy Improvement
 $policy_stable \leftarrow true$
For each $s \in \mathcal{S}$:
 $old_action \leftarrow \pi(s)$
 $\pi(s) \leftarrow \arg \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V(s')]$
If $old_action \neq \pi(s)$, then $policy_stable \leftarrow false$
If $policy_stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Generalized Policy Iteration



Policy iteration consists of two simultaneous, interacting processes, one making the value function consistent with the current policy (policy evaluation), and the other making the policy greedy with respect to the current value function (policy improvement). In policy iteration, these two processes alternate, each completing before the other begins, but this is not really necessary. In value iteration, for example, only a single iteration of policy evaluation is performed in between each policy improvement. In asynchronous dynamic programming methods, the evaluation and improvement processes are interleaved at an even finer grain. In some cases a single state is updated in one process before returning to the other. As long as both processes continue to update all states, the ultimate result is typically the same—convergence to the optimal value function and an optimal policy.

Sutton and Barto, Chapter 4.6

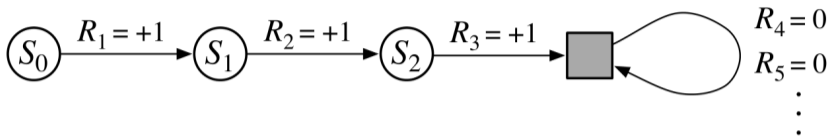
Notes

Standard Q-learning can be seen as an example of asynchronous dynamic programming, where in every step the policy is evaluated only for the current state action pair and the policy is only improved (if necessary) for the current state.



Policy iteration consists of two simultaneous, interacting processes, one making the value function consistent with the current policy (policy evaluation), and the other making the policy greedy with respect to the current value function (policy improvement). In policy iteration, these two processes alternate, each completing before the other begins, but this is not really necessary. In value iteration, for example, only a single iteration of policy evaluation is performed in between each policy improvement. In asynchronous dynamic programming methods, the evaluation and improvement processes are interleaved at an even finer grain. In some cases a single state is updated in one process before returning to the other. As long as both processes continue to update all states, the ultimate result is typically the same—convergence to the optimal value function and an optimal policy.

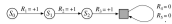
Terminal or Absorbing States



Sometimes we wish to model something in between fixed-horizon and infinite-horizon: settings where an episode can terminate, because some task is completed, independently of *when* this happens.

Such terminal states can be modeled with absorbing states that transition deterministically (and for any action) to themselves, without any immediate reward. This corresponds to fixing the values for absorbing states at 0.

Notes



Sometimes we wish to model something in between fixed-horizon and infinite-horizon settings where an episode can terminate, because some task is completed, independently of when this happens.

Such terminal states can be modeled with absorbing states that transition deterministically (and for any action) to themselves, without any immediate reward. This corresponds to fixing the values for absorbing states at 0.

The planning example in the simplified map of some cities in Europe can be modeled by introducing a self-transition with zero reward for the goal city and removing all out-going transitions, e.g. for Rome. If we want to use the same map to solve another planning problem, we would re-insert the out-going transitions and remove the self-transition for Rome and apply these changes to the new goal city. This modification of the MDP results effectively in fixing the value for the goal city at 0.

Maximizing Future Discounted Values with Linear Programming

As an alternative to dynamics programming, one can define the problem of finding optimal values as a linear program.

Let us use the notation $v_s = V_\gamma^\infty(\pi^*, s)$. For the optimal policy π^* we have

$$v_s \geq r_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{s \rightarrow s'}^a v_{s'}, \forall a \in \mathcal{A}_s, s \in \mathcal{S}. \quad (6)$$

This allows us to define the linear program

$$\min_{v_s} \sum_{s \in \mathcal{S}} v_s \quad (7)$$

subject to

$$v_s \geq r_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{s \rightarrow s'}^a v_{s'}, \forall a \in \mathcal{A}_s, s \in \mathcal{S}$$

Notes

As an alternative to dynamics programming, one can define the problem of finding optimal values as a linear program.

Let us the notation $v = V_{\pi}^*(r, \gamma)$. For the optimal policy π^* we have

$$v_s \geq c_s^* + \gamma \sum_{a \in \mathcal{A}_s} p_{ss'}^a v_{s'} \quad \forall s \in \mathcal{S}. \quad (6)$$

This allows us to define the linear program

$$\max_{v \in \mathbb{R}^{\mathcal{S}}} \sum_{s \in \mathcal{S}} v_s \quad (7)$$

subject to

$$v_s \geq c_s^* + \gamma \sum_{a \in \mathcal{A}_s} p_{ss'}^a v_{s'} \quad \forall s \in \mathcal{S}$$

Linear Programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. There exist efficient solvers for linear programming problems.

We will not discuss linear programming solutions of MDPs in details, but you should know that MDPs can be solved with linear programming and you should get an idea how MDP problems can be described as a linear programming problems.

All techniques to solve MDPs can give us inspirations for solving reinforcement learning problems (see for example <http://proceedings.mlr.press/v130/bas-serrano21a.html>).

Maximizing the Reward Rate with Linear Programming

Let us define the Markov chain with transition probabilities $p_{s \rightarrow s'}^\pi = \sum_{a \in \mathcal{A}_s} \pi(a|s) p_{s \rightarrow s'}^a$ and let us assume that this Markov chain is irreducible and aperiodic for all π . Then there exists the stationary distribution ρ^π . Note that this is a strong assumption, which, however, is e.g. fulfilled when $p_{s \rightarrow s'}^a$ is positive for all a, s, s' . Let us define the reward rate

$$\bar{r} = \sum_{s \in \mathcal{S}, a \in \mathcal{A}_s} \pi_s^\pi \pi(a|s) r_s^a$$

and introduce the variables $c_s^a = \rho_s^\pi \pi(a|s)$. Then we can define the linear program

$$\max_{c_s^a} \sum_{s \in \mathcal{S}, a \in \mathcal{A}_s} r_s^a c_s^a \quad (8)$$

subject to

$$\sum_{s \in \mathcal{S}, a \in \mathcal{A}_s} c_s^a = 1 \text{ and } \sum_{a \in \mathcal{A}_{s'}} c_{s'}^a = \sum_{s \in \mathcal{S}, a \in \mathcal{A}_s} c_s^a p_{s \rightarrow s'}^a \quad (9)$$