Lecture Notes:

Discrete Markov Decision Processes and Utility

Course: Symbolic AI

Written by: Thomas Moerland

Contents

1 Preliminaries: Functions and discrete probability distributions

As a quick recap, we shortly summarize the mathematical notations of: 1) sets, 2) functions, 3) discrete probability distributions, and 4) expectations of discrete probability distributions. We will need these concepts throughout the remainder of these lecture notes.

1.1 Sets

- Discrete set: A set of countable values, for example $\mathcal{X} = \{1, 2, .., n\}$, or $\{\text{up,down,left,right}\}$.
- Continuous set: A set of connected numbers. An example are all numbers in the interval $[2, 11]$, or the real line (denoted by $\mathbb{R}$).

The *cardinality* (size) of a discrete set $\mathcal{X}$ is denoted by $|\mathcal{X}|$. For example, for $\mathcal{X} = \{0, 1, 2\}$, we have $|\mathcal{X}| = 3$.

1.2 Definition of a function

- A function f maps a value in the function's *domain* $\mathcal{X}$ to a (unique) value in the function's *co-domain/range* $\mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ can be discrete or continuous sets.
- We write the statement that f is a function from $\mathcal{X}$ to $\mathcal{Y}$ as $f : \mathcal{X} \to \mathcal{Y}$.

1.3 Discrete probability distribution

- A *discrete variable* X can take values in a discrete set $\mathcal{X} = \{1, 2, .., n\}$. A particular value that X takes is denoted by x.
- Discrete variable X has an associated *probability distribution*: $p(X)$, where $p : \mathcal{X} \to [0, 1]$.
  - Each possible value x that the variable can take is associated with a probability $p(X = x) \in [0, 1]$.
  - For example, $p(X = 1) = 0.2$, i.e., the probability that X is equal to 1 is 20%.
- Probability distributions always sum to 1, i.e.: $\sum_{x \in \mathcal{X}} p(x) = 1$.

Example: A variable X that can take three values ($\mathcal{X} = \{1, 2, 3\}$), with associated probability distribution $p(X = x)$:

$p(X=1)$	$p(X=2)$	$p(X=3)$
0.2	0.4	0.4

Conditional distributions

- A conditional distribution means that the distribution of one variable depends on the value that another variable takes.
- We write $p(X|Y)$ to indicate that the value of X depends on the value of Y. For discrete random variables, we may store a conditional distribution as a table of size $|\mathcal{X}| \times |\mathcal{Y}|$.

Example: A variable X that can take three values ($\mathcal{X} = \{1,2,3\}$) and variable Y that can take two values ($\mathcal{Y} = \{1,2\}$). The conditional distribution may for example be:

	$p(X=1\|Y)$	$p(X=2\|Y)$	$p(X=3\|Y)$
$Y=1$	0.2	0.4	0.4
$Y=2$	0.1	0.9	0.0

Note that for each value Y, $p(X|Y)$ should still sum to 1, i.e., it is a valid probability distribution. In the table above, each row therefore sums to 1.

1.4 Expectation (of a function of a discrete variable)

We also need the notion of an *expectation.* The expectation of a random variable is essentially an average. In these notes, we will need the average *of a function of the random variable*, denoted by $f(\cdot)$.

- Assume a function $f : \mathcal{X} \rightarrow \mathbb{R}$, which, for every discrete value $x \in \mathcal{X}$ maps to a continuous value $f(x) \in \mathbb{R}$.

  Example:

x	$p(X=x)$	$f(x)$
1	0.2	22.0
2	0.3	13.0
3	0.5	7.4

- The expectation is then defined as follows:

$$\mathbb{E}_{X\sim p(X)}[f(X)] = \sum_{x\in X} [f(x) \cdot p(x)] \tag{1}$$

The formula may look complicated, but it essentially reweights each function outcome by the probability that this output occurs. It is therefore trivial to compute the expectation of the above example:

Example:

$$\begin{aligned}\mathbb{E}_{X\sim p(X)}[f(X)] &= 22.0 \cdot 0.2 + 13.0 \cdot 0.3 + 7.4 \cdot 0.5 \\ &= 12.0\end{aligned}$$

2 Definition of Discrete MDP

A discrete Markov Decision Process is a very generic way to define sequential decision-making tasks. Discrete refers to the fact that the the states and actions are *atomic*. Compared to other planning problem definitions, MDPs:

- can handle multiple goals (through utility).
- includes stochastic environments.

MDP formulations for example include all shortest-path problems.

2.1 Formal definition

The formal definition of a MDP consists of six elements:

1. **State space**: $\mathcal{S}$.
   - Which observations are possible?
   - *Discrete set* of size $|\mathcal{S}|$.
2. **Action space**: $\mathcal{A}$.
   - Which actions are possible?
   - *Discrete set* of size $|\mathcal{A}|$.
3. **Transition dynamics**: $T(s'|s,a)$.
   - How does the environment react to an action?
   - *A conditional probability distribution*, represented as a *function* $T : \mathcal{S} \times \mathcal{A} \to p(\mathcal{S})$.
   - Represented as table of size $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$, each entry a probability.
   - **Terminal states**: some states are terminal. When we reach them, the task ends (i.e., we cannot select a new action, or transition out of them.
4. **Reward function**: $R(s,a,s')$.
   - How rewarding/preferable is each transition?
   - *Function* $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.
   - Represented as table of size $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$, each entry a real number (the associated reward).
   - Frequently simpler functions, like like $R(s')$ or $R(s,a)$.

- *Cost*, as frequently used in shortest path-problems, is equivalent to *negative reward.* We can formalize all shortest path problems as an MDP. The only difference between both specification is that we want to minimize cost, while in MDPs we want to maximize reward. When we negate the cost function, we can simply solve a shortest path problem as an MDP.

  Example: There a two paths from A to B, the first has cost 5, the second cost 8. We prefer the first path, since it has the smallest cost. Reformulated, the first path has a reward of -5, and the second a reward of -8. When we now optimize reward, we still prefer the first path.

5. **Discount factor**: γ.
   - How much do we down-weight long-term rewards?
   - A *constant*, $\gamma \in [0, 1]$.
6. **Initial state distribution**: $p_0(s)$.
   - Where do we start?
   - A *distribution* over $\mathcal{S}$.
   - Represented as a table of size $|\mathcal{S}|$.

Notation convention

- At each timestep we observe s_t and take action a_t, after which we observe $s_t \sim T(\cdot|s, a)$.
- Shorthand notation for reward with time index: $r_t = R(s_t, a_t, s_{t+1})$.
- Sometimes we drop the time-index. A single transition is then written as $\{s, a, r, \text{s'}\}$, for an arbitrary transition (without time specification). The next state s' is pronounced as '*s prime*'.

2.2 Example of MDP definition

To illustrate the definition of a MDP, we use the below task, which also appears in the lecture.

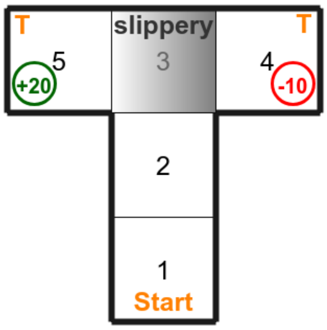


Figure 1: Example Markov Decision Process

Description of task in words

- There are 5 observable states, as numbered in the figure.
- In each state, we have four possible actions: up, down, left, right.
- Each action moves the agent in that direction. When we hit a wall, the agent stays in the same location. When we step on state 3 (a 'slippery' state), we have 20% chance of accidently slipping to state 4. States 4 and 5 are terminal: when we reach them, the task ends.
- When we reach state 5 we get a reward of +20, while reaching state 4 gives a penalty of -10. All other transitions have a small penalty of -1.
- We care equally much about distant future rewards as about immediate rewards.
- We always start in state 1.

Formal definition of this task in MDP terminology

- State space: $\mathcal{S} = \{1, 2, 3, 4, 5\}$
- Action space: $\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\}$
- Transition dynamics:

s	**a**	$T(s'=1)$	$s'=2$	$s'=3$	$s'=4$	$s'=5$
1	up	0.0	1.0	0.0	0.0	0.0
1	down	1.0	0.0	0.0	0.0	0.0
1	left	1.0	0.0	0.0	0.0	0.0
1	right	1.0	0.0	0.0	0.0	0.0
2	up	0.0	0.0	0.8	0.2	0.0
2	down	1.0	0.0	0.0	0.0	0.0
etc.	..	..	..	..	..	..

- Reward function:

s'	$R(s')$
1	-1.0
2	-1.0
3	-1.0
4	-10.0
5	+20.0

- Discount factor: $\gamma = 1.0$
- Initial state distribution:

s	$p_0(s)$
1	1.0
2	0.0
3	0.0
4	0.0
5	0.0

3 Policy

A policy specifies how we act in the environment, i.e., which action we take in each state. Formally, it is a *function* specifying the probability of each action in each state.

- Policy: $\pi(a|s)$, where $\pi : \mathcal{S} \to p(\mathcal{A})$.
  - In a discrete MDP (state and action space are discrete), this can be stored as a table of size $|\mathcal{S}| \times |\mathcal{A}|$.
  - Stationary/time-independent, i.e., same function for all timepoints.

Example:

s	$\pi(a{=}\text{up}|s)$	$\pi(a{=}\text{down}|s)$	$\pi(a{=}\text{left}|s)$	$\pi(a{=}\text{right}|s)$
1	0.2	0.8	0.0	0.0
2	0.0	0.0	0.0	1.0
3	0.7	0.0	0.3	0.0
etc.	..	..	..	..

3.1 Deterministic policy

A special case of a policy is a *deterministic policy*, denoted by $\pi(s)$. A deterministic policy selects only a single action in every state. Of course the deterministic action may differ between states, as in the below example:

Example:

s	$\pi(a{=}\text{up}|s)$	$\pi(a{=}\text{down}|s)$	$\pi(a{=}\text{left}|s)$	$\pi(a{=}\text{right}|s)$
1	0.0	1.0	0.0	0.0
2	0.0	0.0	0.0	1.0
3	1.0	0.0	0.0	0.0
etc.	..	..	..	..

- A key example of a deterministic policy is the *greedy* policy, which selects the best available action. We will get back to this topic later in the notes.
- Second, the above example *explicitly* stores the policy as a table of probabilities. However, we can also *implicitly* store the policy, in the form of a value table/function. We will get back to this topic later in the lecture notes as well.

4 Cumulative reward and value (utility)

We have not yet defined what we actually want to achieve in the sequential decision-making task. The MDP definition included a reward function. We clearly want to achieve as much reward as possible in the task. The sum of all the reward that we achieve is known as the *cumulative reward*, also called the *return*. Since the environment can be stochastic, and our policy can be stochastic, we are really interested in the return that we achieve *on average*. The average return is known as the *value* (also referred to as the *utility*). We will formally define these concepts below.

4.1 Return/cumulative reward

- **Trace**: When we repeatedly act in the MDP, we generate a *trace* of states, actions, rewards and next states. We denote such a trace, starting at timestep t, of length n, by

  $$h_t = \{s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, ..., a_{t+n}, r_{t+n}, s_{t+n+1}\}.$$

  Example: A short trace of length 3 could look like:

  $$\{s_0\text{=}1, a_0\text{=up}, r_0\text{=-}1, s_1\text{=}2, a_1\text{=up}, r_1\text{=-}1, s_2\text{=}3, a_2\text{=left}, r_2\text{=}20, s_3\text{=}5\}$$

- **Return of trace**: We can now define the cumulative reward of a trace h_t as:

  $$\begin{aligned} G_t &= r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + ... + \gamma^n \cdot r_{t+n} \\ &= \sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} \end{aligned} \tag{2}$$

  We need to specify the *discount factor* $\gamma \in [0, 1]$. It determines how much we downweight distant rewards. We may pick γ ourselves, but two extremes are:

  - $\gamma = 0$: A myopic agent, which only considers the immediate reward, i.e., $G_t = r_t$.
  - $\gamma = 1$:[1] A far-sighted agent, which treats all future rewards as equal, i.e., $G_t = r_t + r_{t+1} + r_{t+2} + ...$

[1]Formally, we cannot set $\gamma = 1.0$ when we use an infinite-horizon return as defined in Eq. 2, because the cumulative reward may also become infinite (if the task can indeed continue forever, for example when there is a loop with non-zero total reward). We choose to ignore these issues for these notes, and in most of our experiments simply use $\gamma = 1.0$, which weights all rewards equally, whether nearby or far-away.

Example: We use the earlier trace example, and assume $\gamma = 0.9$. The cumulative reward is equal to:

$$G_0 = -1 + 0.9 \cdot -1 + 0.9^2 \cdot 20 = 16.2 - 1.9 = 14.3$$

4.2 Values

The return of a trace is not the real measure of optimality in which we are interested. The environment can be stochastic, and our policy as well, so for a given policy we do not always observe the same trace. Therefore, we are actually interested in the *average*, or expected, cumulative reward that a certain policy achieves. The average cumulative reward is better known as the *value.* We can define *state values*, and *state-action values.*

4.3 State value

- We define the *state value* $V(s)$, which is the *average* return we expect to achieve when an agent starts in state s and follows policy π, as:

$$V^{\pi}(s) = \mathbb{E}_{\pi,T}\big[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s\big] \tag{3}$$

- See Sec. 1 for the definition of an expectation $\mathbb{E}[\cdot]$. Here, $\mathbb{E}_{\pi,T}$ means that we take the expectation over all possible traces induced by the policy π and dynamics T. So, we essentially compute the return of all possible traces, and reweight each return by the probability that it will occur given π and T.

  Example: Imagine we have a policy π, which from state s can result in two traces. The first trace has a cumulative reward of 20, and occurs in 60% of the times. The other trace has a cumulative reward of 10, and occurs 40% of the times. *What is the value of state s?*

$$V^{\pi}(s) = 0.6 \cdot 20 + 0.4 \cdot 10 = 16.$$

  Note that 16 is the average return (cumulative reward) that we expect to get from state s under this policy.

- The state value is a *function* $V : \mathcal{S} \to \mathbb{R}$. For every state there is one associated value.

- The state value can be represented as a table of size $|\mathcal{S}|$.

  Example:

s	$V^\pi(s)$
1	2.0
2	4.0
3	1.0
etc.	etc.

- Every policy π has only one unique associated value function $V^\pi(s)$. We sometimes omit π to simplify notation, simply writing $V(s)$, knowing a state value is always conditioned on a certain policy.

- **The state value of a terminal state is by definition zero**, i.e.,

$$s = \text{terminal} \quad \Rightarrow \quad V(s) := 0.$$

4.4 State-action value

Instead of state values $V^\pi(s)$, we also define state-action values $Q^\pi(s,a)$. The only difference is that we now condition on a state *and action*, i.e., we estimate the average return we expect to achieve when taking action a in state s, and then following policy π afterwards:

$$Q^\pi(s,a) = \mathbb{E}_{\pi,T}\big[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s, a_t = a\big] \tag{4}$$

- The state-action value is a *function* $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. For every state-action pair it maps to a real number. It can be represented as a table of size $|\mathcal{S}| \times |\mathcal{A}|$.

  Example: Each table entry stores a $Q(s,a)$ estimate for the specific s, a combination:

	a=up	a=down	a=left	a=right
s=1	4.0	3.0	7.0	1.0
s=2	2.0	-4.0	0.3	1.0
s=3	3.5	0.8	3.6	6.2
etc.	..	..	..	..

- Every policy π has only one unique associated state-action value function $Q^\pi(s,a)$.

- **The state-action value of a terminal state is by definition zero**, i.e.,

$$s = \text{terminal} \quad \Rightarrow \quad Q(s,a) := 0, \quad \forall a$$

- Again, every policy π has only one unique associated state-action value function $Q^{\pi}(s,a)$. We sometimes omit π to simplify notation, simply writing $Q(s,a)$, knowing a state-action value is always conditioned on a certain policy.
- A potential benefit of state-action values (Q) versus state values (V) is that state-action values directly tell what every action is worth (which may be useful for action selection). However, the state-action table is larger to store than the state table.

5 Optimal value function and optimal policy

Given a MDP we are of course interested in the best policy. The best policy is defined in terms of the value. We want to find the policy which has the highest value, i.e., the policy which on average gives the highest cumulative pay-off from all states.

5.1 Partial ordering

We first need a way to define when one value function is 'better' than another value function. For two numbers, it is straightforward to idenfity which one is higher than the other. But how do we identify when an entire function is higher than another?

For this means, we introduce a *partial ordering* over value functions. For two value functions π and π', with associated value functions $V^{\pi}(s)$ and $V^{\pi'}(s)$, we say:

$$\pi' \geq \pi \quad \Leftrightarrow \quad V^{\pi'}(s) \geq V^{\pi}(s) \quad \text{for all } s \in \mathcal{S} \tag{5}$$

In words, π' is better than π if and only if its associated value estimates are a least equally good *at all states of the environment.*

5.2 Optimal value function

Every possible policy in the MDP leads to one associated value function. Interestingly, it turns out that of all these possible value functions in the MDP, there is only one optimal one.

Theorem: For a given MDP, there is only one optimal value function, denoted by $V^{\star}(s)$:

$$V^{\star}(s) = \max_{\pi} V^{\pi}(s) \tag{6}$$

For this optimal value function, we naturally have:

$$V^{\star}(s) \geq V^{\pi}(s), \quad \text{for all } \pi \in \Pi, s \in \mathcal{S} \tag{7}$$

- In a given MDP, there is only one optimal value function.

5.3 Optimal policy

The policy that achieves the optimal value function is called an *optimal policy*, denoted by $\pi^\star$:

$$\pi^\star(s) = \arg\max_\pi V^\pi(s) = \arg\max_\pi Q^\pi(s,a) \tag{8}$$

- There may be multiple policies that achieve the optimal value function, but we simply denote all of them by $\pi^\star$.
- In a MDP the optimal policy is always *greedy*. In words, in each state there is always one action which is best. (Unless there are two or more actions available with exactly the same value: in that case, we can select either of them).

Example: In the example MDP from Sec. 2.2, the optimal policy is:

s	$\pi^\star(s)$
1	up
2	up
3	left

Imagine shortly we remove the stochasticity in the environment of Sec. 2.2. We can reason what the optimal value and state-action values should be (what is the best return we can achieve from a particular state or state-action).

s	$V^\star(s)$
1	18
2	19
3	20

	Q(s,a=up)	Q(s,a=down)	Q(s,a=left)	Q(s,a=right)
s=1	18	16	17	17
s=2	19	17	18	18
s=3	19	18	20	-10

Once again, these are the optimal values without stochasticity. Look at the environment again and see if you understand the above optimal values, when you try to reason what the highest cumulative reward is that we can achieve from a certain state or state-action. See the lecture for computations with stochasticity.

5.4 Goal op MDP optimization

The goal op MDP optimization is to find (an) optimal policy $\pi^{\star}(s)$. We can also search for the optimal value function $V^{\star}(s)$ or $Q^{\star}(s,a)$, because the optimal value function will directly gives us the optimal policy (see Sec. 7.5). However, we first need to discuss the Bellman equation.

6 Bellman equation

The interesting thing about the value function is that we can write it in *recursive form*, because the value is also defined at the next states.

6.1 Bellman equation for state values (V(s))

The Bellman equation for state values $V(s)$ is given by:

$$\begin{aligned} V(s) &= \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \big[r + \gamma \cdot V(s') \big] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \Big[\sum_{s' \in \mathcal{S}} T(s'|s,a) \big[r + \gamma \cdot V(s') \big] \Big] \end{aligned} \tag{9}$$

Note that the expectations do no longer run over all traces, but only over the first action (a) and transition (s'). In the bottom equation, we explicitly wrote out the expectations to show the summations.

Back-up diagrams We can graphically illustrate the above equation using a *back-up diagram*. The back-up diagram for the above equation is shown in Figure 2. In these figures, white nodes represent states, and black nodes represent actions. The Bellman equation gives a relation between the value at a state s and the values at the possible next states s'. To compute the value at state s, we should sum over all the available actions (2 in the diagram), and for each action sum over all the available next states (for each action there are two possible next states in the diagram).

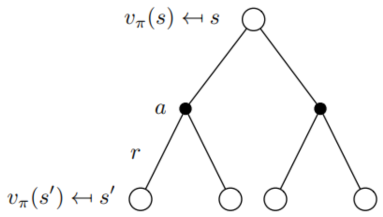

$v_\pi(s) \leftarrow\!\shortmid s$
a
r
$v_\pi(s') \leftarrow\!\shortmid s'$

Figure 2: Graphical illlustration of the Bellman equation for $V^\pi(s)$. The Bellman equation describes a relation between the state value at s, and the state value at its possible decessor states s'.

Example: We will use Figure 2 as an example, so two actions, and two next states for each action. For the policy, we have 0.2 chance to take the left action, and 0.8 chance to take the right action. For each action, we have 0.5 chance to transition to each of the possible next states. The rewards on all these transitions are 1. The value estimates at the possible next states s' (from left to right) are 2, 4, 6, and 8, respectively. Use $\gamma = 0.8$.

Compute $V(s)$ according to the Bellman equation:

$$\begin{aligned} V(s) =& 0.2 \cdot (0.5(1+0.8\cdot 2) + 0.5(1+0.8\cdot 4)) + \\ & 0.8 \cdot (0.5(1+0.8\cdot 6) + 0.5(1+0.8\cdot 8) \\ =& 0.2 \cdot 3.4 + 0.8 \cdot 6.6 \\ =& 5.96 \end{aligned}$$

6.2 Bellman equation for state-action values (Q(s,a))

The same recursion principle applies to state-action values, which gives the Bellman equation:

$$\begin{aligned} Q(s,a) &= \mathbb{E}_{s'\sim T}\big[r + \gamma \cdot \mathbb{E}_{'a\sim\pi}[Q(s',a')]\big] \\ &= \sum_{s'\in\mathcal{S}} T(s'|s,a)\big[r + \gamma \cdot \sum_{a'\in\mathcal{A}} [\pi(a|s)\cdot Q(s',a')]\big]. \end{aligned} \tag{10}$$

We again have to take an expectation over actions, and and expectation over the dynamics, but the order is reversed when we write Q-values. The back-up diagram for the Bellman equation with Q-values is given in Figure 3. Compare this figure to Figure 2.

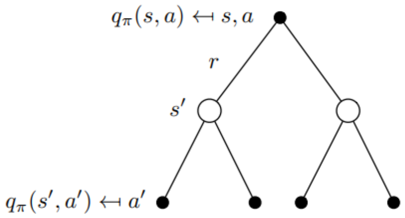

$q_\pi(s,a) \leftarrow\!\!\!\shortmid s, a$
r
s'
$q_\pi(s',a') \leftarrow\!\!\!\shortmid a'$

Figure 3: Graphical illustration of the Bellman equation for Q-values (Eq. 10).

Example: We will use Figure 3 as an example. We want to compute $Q^\pi(s,a)$. We may transition to two next states, with probability (from left to right) of 0.4 and 0.6 and rewards of 2 and 4, respectively. In each next state, we have two available actions. The current estimates of these actions are 4, 3, 2, 1 (from left to right). Our policy is greedy, i.e., we always select the action with the highest value estimate. Use $\gamma = 1.0$.

Compute $Q(s,a)$ according to the Bellman equation:

$$\begin{aligned} Q(s,a) =& 0.4 \cdot (2 + 1.0 \cdot (1.0 \cdot 4 + 0.0 \cdot 3))+ \\ & 0.6 \cdot (4 + 1.0 \cdot (1.0 \cdot 2 + 0.0 \cdot 1)) \\ =& 0.4 \cdot 6.0 + 0.6 \cdot 6.0 \\ =& 6.0 \end{aligned}$$

Compared to the $V(s)$ representation, we essentially switched the nodes at which we represent the solution (states or state-actions). Each policy still has one associated value function, but we can represent this value function in two ways in memory.

The recursive nature of the Bellman equation is crucial, because most solution algorithms use this recursive nature in their solution algorithms. We will look at a key example in the next section: Dynamic Programming.

6.3 Relation between V(s) and Q(s,a)

The Bellman equation also allows us to get better insight into the relation between state values $V(s)$ and state-action values $Q(s,a)$. There is only one true value function given a policy, and state values or state-action values are just different ways of representing this function. We can therefore also rewrite both into eachother.

- **From Q to V**: Given the state-action values and the policy, we can compute the value of a state as:

$$\begin{aligned} V(s) &= \mathbb{E}_{a\sim\pi}[Q(s,a)] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \cdot Q(s,a) \end{aligned} \tag{11}$$

  The back-up diagram is shown in Figure 4. We essentially reweight each state-action value estimate by its probability under the policy.

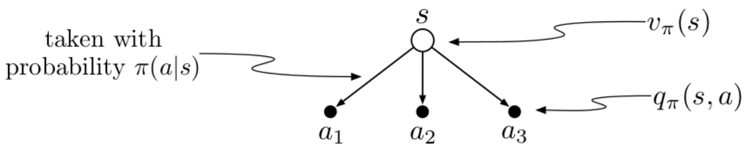


Figure 4: From Q to V. Illustration of Eq. 11.

Example: In a state s we have four actions, with value estimates $Q(s,1) = 1$, $Q(s,2) = 2$, $Q(s,3) = 5$ and $Q(s,3) = 4$.

– *We follow a random policy. What is the state value?*

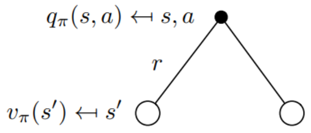
$$V(s) = (1/4) \cdot 1 + (1/4) \cdot 2 + (1/4) \cdot 5 + (1/4) \cdot 4 = 3.$$

– *Instead, we were using a greedy policy. What is the state value?*
The greedy policy puts all probability at the action with the highest value estimate. So:

$$V(s) = 0.0 \cdot 1 + 0.0 \cdot 2 + 1.0 \cdot 5 + 0.0 \cdot 4 = 5.$$

- **From V to Q**: We can also compute state-action values from state values, according to:

$$\begin{aligned} Q(s,a) &= \mathbb{E}_{s' \sim T}[r + \gamma V(s')] \\ &= \sum_{s' \in \mathcal{S}} T(s'|s,a)[r + \gamma V(s')]. \end{aligned} \tag{12}$$



$q_\pi(s,a) \leftarrow\!\shortmid\ s,a$
r
$v_\pi(s') \leftarrow\!\shortmid\ s'$

Figure 5: From V to Q. Illustration of Eq. 12.

Substituting both equations into eachoter Eq. 11 and 12 give the relations between $V(s)$ and $Q(s, a)$. Note that when we substitute both into eachother, we directly retrieve the Bellman equation.

- Substitute the expression for $Q(s, a)$ in Eq. 12 into Eq. 11, and you get the Bellman equation for state values (Eq. 9).
- Try it yourself the other way around (substitute the expression for $V(s)$ in Eq. 11 into Eq. 12), and retrieve the Bellman equation for Q-values (Eq. 10).

7 Dynamic Programming

We our now ready to look at an actual solution method for the MDP. A key approach, on which many other algorithms are built, is *dynamic programming* (DP).

Dynamic Programming is a general principle, in which we break down a larger problem into smaller subproblems, which are more easy to solve.

However, from now on, when you mention Dynamic Programming, we mean Dynamic Programming (DP) for solving MDPs.

Strengths

- Dynamic Programming is a generic approach to solve a MDP.
- It requires no heuristic.
- It is guaranteed to find the optimal solution.

Weaknesses

- It requires us to store all states (the entire state space).
- Sweeps through the state space can be inefficient.

DP is the basis for many other approaches to solve MDPs, like reinforcement learning.

Two procedures At the core of DP is the iteration of two procedures:

- Policy evaluation ($\pi \to V/Q$): given a policy π, how do we compute the value $V^{\pi}(s)$ or $Q^{\pi}(s,a)$.
- Policy improvement ($V/Q \to \pi'$): given a value function $V^{\pi}(s)$ or $Q^{\pi}(s,a)$, how do we compute *a better policy* π.

The key idea will be to iterate both procedures. This is graphically illustrated in Figure 6. On the left we iterate $\pi \to V^{\pi}$ (policy evaluation) and $V^{\pi} \to$ newπ (policy improvement). On the right you see that this procedure eventually converges on the optimal policy $\pi^{\star}$ and value function $V^{\star}$ or $Q^{\star}$. We will detail these procedures in more detail below.

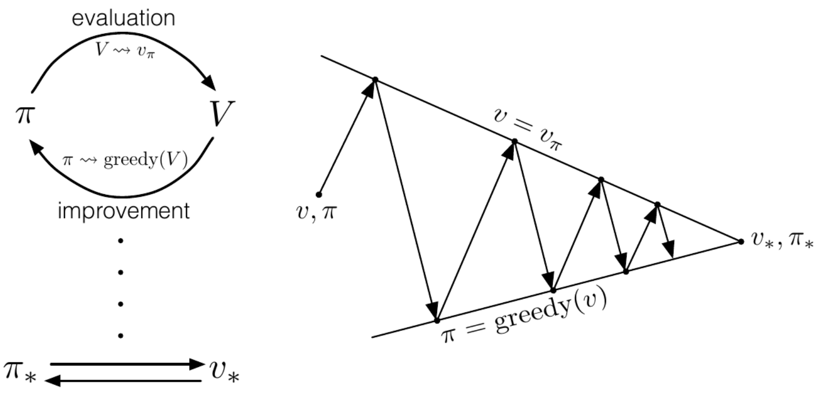


Figure 6: Graphical illustration of dynamic programming, in which we alternate policy evaluation and policy improvement. Right: alternating these two procedures generates a contraction mapping to the optimal value and policy functions.

7.1 Policy evaluation

Our goal in policy evaluation is to find the true value function $V^\pi(s)$ of a particular policy π. The DP approach to this problem is:

1. Initialize a value table $V(s)$ (or $Q(s,a)$). All entries can be random, except for all terminal states, which have $V(s = \textit{terminal}) = 0$ or $Q(s = \textit{terminal}, a) = 0$ by definition.

2. Loop through all states s (or state-actions (s,a)), each time evaluating the Bellman equation to update the specific state.

3. Repeat until all values converge. The converged estimates are $V^\pi(s)$ (or $Q^\pi(s,a)$), the value function of policy π.

The policy evaluation algorithm for state values is shown in Algorithm 1. It takes as input a policy and acceptance threshold, and outputs the value function for the specific policy. It sweeps through the entire state space, at each state evaluating the Bellman equation. The other lines of the algorithms are only checking whether we have already converged, i.e., whether our maximum error in an iteration is below a certain acceptance threshold.

7.2 Policy improvement

We now know how to compute the value function for a given policy. But how may we improve the policy, given that we know its value function. A crucial insight, as already discussed before, is that

in the MDP specification, the optimal policy is always globally greedy.

In other words, in every state of the problem, there is always one action which is best (or multiple actions which are all equally good, so it does not matter which one we take). This is similar to ideas in classic search, where we usually aim to find a single, optimal trajectory.

Since we aim to find the greedy optimal policy, the policy improvement step in Dynamic Programming can also be greedy. The idea is to let the new policy be greedy with respect to the current value function estimate. We simply put all probability mass at the action which currently has the highest value estimate in a state. We may identify the greedy action as:

$$\pi(s) \leftarrow \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \big[r + \gamma \cdot V(s') \big], \tag{13}$$

or, if we learned state-action values, as

$$\pi(s) \leftarrow \arg\max_a Q(s,a). \tag{14}$$

Notice that, when we store $Q(s,a)$ estimates, then policy improvement is really easy: we simply look-up the values in the table (i.e., slicing the row of the relevant state in the Table example of Sec. 4.4). When we store state values, then policy improvement requires us to evaluate the the Bellman equation once more.

The above updates may change the policy at certain states, leading to a new policy π'. We can then start a new round of policy evaluation, trying to find the value function for this new policy. This starts the DP loop, which we further detail below.

7.3 Dynamic Programming

We now understand which two procedures DP will iterate. There is one crucial decision left, which leads to two variants of Dynamic Programming. Policy evaluation may require many iterations to converge. We can wait until convergence before we do policy improvement, but this may expend much budget to let policy evaluation converge, while the right policy improvement is already clear. Therefore, as an alternative, we can also do only a single sweep of policy evaluation (i.e., partial policy evaluation), and then already do policy improvement. This is also guaranteed to converge at the optimal policy.

- **Policy iteration**: We do full policy evaluation until convergence, after which we do policy improvement.
- **Value iteration**: We do one sweep through the state space of policy evaluation, after which we do policy improvement.

7.4 Policy Iteration

Algorithm 2 shows a full description of policy iteration (using $V(s)$). You do not need to implement this algorithm yourself, but you do need to conceptually understand it.

- We will repeat policy evaluation and policy improvement until we have converged (and C remains `True` after policy improvement).
- In this loop, we first do policy evaluation, as per Algorithm 1.
- Next, we do policy improvement, where we greedily improve the policy at every state.
- When the policy no longer changes, we have converged at the optimal policy.

7.5 Implicit policies from value tables

Before we can discuss Value Iteration (VI), we first need to discuss another key idea. So far, we have explicitly stored the policy as a table with probabilities of actions. However, we can also *implicitly* store the policy in the value table. In that case, we only need to store a value table, and can directly derive the policy from the values. For example, we can specify the greedy policy as:

$$\pi(s) = \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right] \tag{15}$$

$$\tag{16}$$

or, when we use a state-action value table, as

$$\pi(s) = \arg\max_a Q(s, a|\pi) \tag{17}$$

Example: Which greedy policy does the following state-action value table implicitly represent?

s	$Q^\pi(s, a = \text{up})$	$Q^\pi(s, a = \text{down})$	$Q^\pi(s, a = \text{left})$	$Q^\pi(s, a = \text{right})$
1	4.0	5.0	3.0	2.0
2	6.0	1.0	2.5	7.5
3	7.5	3.0	3.0	-2.0

Answer: $\pi(s = 1) = \text{left}$, $\pi(s = 2) = \text{up}$, $\pi(s = 3) = \text{right}$ (Or, when represented as a full table, the greedy policy equals the deterministic policy example from Section 3).

7.6 Value Iteration

Value iteration (VI) is the last algorithm we discuss. In VI, we do only one sweep of policy evaluation before we do policy improvement. Moreover, we choose to *implicitly* represent the policy in the form of a value table. The combination of these two idea makes that we can write *policy evaluation and policy improvement in a single equation.* We implement the Bellman equation, but we directly max over the actions. This back-up is better known as the **Bellman optimality operator**, and the underlying equation as the **Bellman optimality equation**.

The relevant equations are for

- Value iteration (on $V(s)$):

$$\begin{aligned} V(s) &\leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \big[r + \gamma \cdot V(s')\big] \\ V(s) &\leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a) \big[r + \gamma \cdot V(s')\big] \qquad (18) \end{aligned}$$

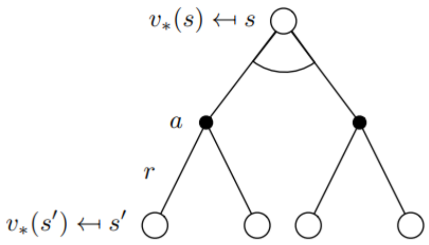


Figure 7: Value iteration back-up (Eq. 18).

This equation is graphically illustrated in Figure 7.

- Q-value iteration (on $Q(s,a)$):

$$\begin{aligned} Q(s,a) &\leftarrow \mathbb{E}_{s' \sim T}\big[r_t + \gamma \max_a Q(s',a')]\big] \\ &\leftarrow \sum_{s' \in \mathcal{S}} T(s'|s,a)\big[r_t + \gamma \max_a Q(s',a')]\big] \end{aligned} \tag{19}$$

This equation is graphically illustrated in Figure 8.

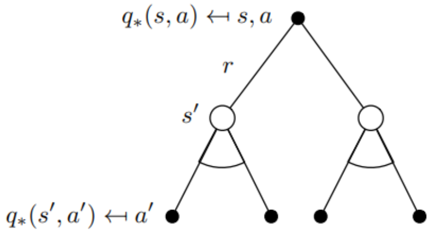

$q_*(s,a) \leftarrow\!\!\dashv s, a$
r
s'
$q_*(s',a') \leftarrow\!\!\dashv a'$

Figure 8: Q iteration back-up (Eq. 19).

With these equation, the value iteration and q-value iteration algorithms become really simple, and only a small modification of the policy evaluation algorithm. The relevant algorithms are shown in Algorithm 3 and 4. Study these algorithms, and see if you understand them, also in relation to the above back-up diagrams. Compare value iteration (VI) to policy iteration (PI): which algorithm is easier to implement?

Algorithm 1: Policy evaluation

```
Input: Policy $\pi(a|s)$, small threshold $\theta$
Result: Value function $V^\pi(s)$
Initialization: A value table $V(s)$ with arbitrary entries, except
$V(terminal) = 0$ ;
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do
        $x \leftarrow V(s)$                                   /* Old value for $s$ */
        $V(s) \leftarrow \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[ r_t + \gamma \cdot V(s') \right]$
        $\Delta \leftarrow \max(\Delta, |x - V(s)|)$           /* Update the max error */
    end
until $\Delta < \theta$;
Return $V(s)$
```

Algorithm 2: Policy iteration

```
Input: Small threshold $\theta$
Result: The optimal greedy policy $\pi^\star(s)$
Initialization: A value table $V(s)$ and policy table $\pi(s)$ with arbitrary
entries, except $V(s = \textit{terminal}) = 0$.
repeat
    /* Policy evaluation                                          */
    repeat
        $\Delta \leftarrow 0$
        for each $s \in \mathcal{S}$ do
            $x \leftarrow V(s)$
            $V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[ r + \gamma \cdot V(s') \right]$
            $\Delta \leftarrow \max(\Delta, |x - V(s)|)$
        end
    until $\Delta < \theta$;
    /* Policy improvement                                         */
    $C = \textsc{True}$
    for each $s \in \mathcal{S}$ do
        $y \leftarrow \pi(s)$                          /* Old policy action */
        $\pi(s) \leftarrow \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[ r + \gamma \cdot V(s') \right]$
        if $\pi(s) \neq y$ then
            $C = \textsc{False}$     /* Policy changed, not converged yet */
        end
    end
until $C = \textsc{True}$;
Return $\pi(s)$
```

Algorithm 3: Value iteration

```
Input: Some deterministic policy $\pi(s)$, small threshold $\theta$
Result: The optimal greedy policy $\pi^\star(s)$
Initialization: A value table $V(s)$ with arbitrary entries, except
$V(s = \textit{terminal}) = 0$
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do
        $x \leftarrow V(s)$
        $V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \big[r + \gamma \cdot V(s')\big]$          /* Eq. 18 */
        $\Delta \leftarrow \max(\Delta, |x - V(s)|)$
    end
until $\Delta < \theta$;
$\pi^\star(s) = \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \big[r + \gamma \cdot V(s')\big] \quad \forall s \in \mathcal{S}$
Return $\pi^\star(s)$
```

Algorithm 4: Q-value iteration

```
Input: Some deterministic policy $\pi(s)$, small threshold $\theta$
Result: The optimal greedy policy $\pi^\star(s)$
Initialization: A state-action value table $Q(s,a)$ with arbitrary entries,
except $Q(s = terminal, a) = 0, \forall a$
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do
        for each $a \in \mathcal{A}$ do
            $x \leftarrow Q(s,a)$
            $Q(s,a) \leftarrow \mathbb{E}_{s' \sim T}\big[r_t + \gamma \max_{a'} Q(s',a')\big]$          /* Eq.  19 */
            $\Delta \leftarrow \max(\Delta, |x - Q(s,a)|)$
        end
    end
until $\Delta < \theta$;
$\pi^\star(s) = \arg\max_a Q(s,a) \quad \forall s \in \mathcal{S}$
Return $\pi^\star(s)$
```