CMPSC 448: Machine Learning

Lecture 17. Markov Decision Processes

Rui Zhang

Fall 2021

PennState

Outline

Markov Decision Processes (MDP)

- Markov Processes
- Markov Reward Processes (MRP)
- Markov Decision Processes (MDP)

MDP is an idealized form of the AI problem for which we have precise theoretical results.
MDP formally describes an environment for RL.
Introduce key components of the mathematics: value functions, policies, and Bellman equations

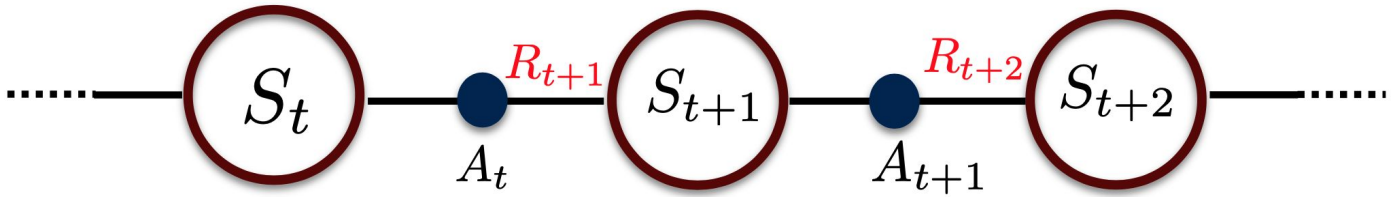
Sequential Decision Making

Agent and environment interact at discrete time steps: $t = 0, 1, \ldots,$

Agent observes state at time step t: $S_t \in \mathcal{S}$

Agent decides an action at time step t: $A_t \in \mathcal{A}(S_t)$

Environment responds with a reward $R_{t+1} \in \mathbb{R}$
and
resulting next state $S_{t+1} \in \mathcal{S}$



The agent-environment interface

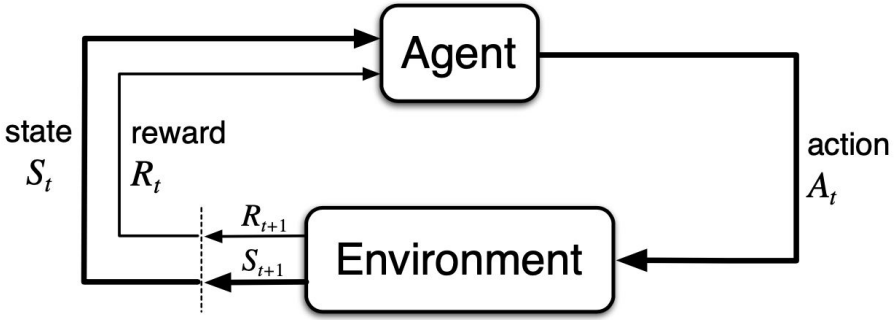


Figure 3.1: The agent–environment interaction in a Markov decision process.

We use R_{t+1} instead of R_t to denote the reward due to A_t because it emphasizes that the next reward and next state, R_{t+1} and S_{t+1}, are jointly determined. Unfortunately, both conventions are widely used in the literature.

The MDP and agent together thereby give rise to a sequence or trajectory that begins like this:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Markov property

“The future is independent of the past given the present”

Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

The state captures all relevant information from the history

Once the state is known, the history may be thrown away

i.e. The state is a sufficient statistic of the future

“Markov” generally means that given the present state, the future and the past are independent

State Transition Matrix

For a Markov state S and successor state s', the state transition probability is defined by

$$P_{s,s'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$$

State transition matrix P defines transition probabilities from all states to all successor states:

$$\boldsymbol{P} = \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} & \dots & P_{1,n} \\ P_{2,1} & P_{2,2} & P_{2,3} & \dots & P_{2,n} \\ \dots & \dots & \dots & \dots & \dots \\ P_{n,1} & P_{n,2} & P_{n,3} & \dots & P_{n,n} \end{bmatrix}$$

n is the # of states

Markov Process (Markov Chain)

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, S_3, \ldots$ with the Markov property

A Markov Process (Markov Chain) is a tuple $\langle \mathcal{S}, \boldsymbol{P} \rangle$

$\mathcal{S}$ is a (finite) set of states

$\boldsymbol{P}$ is a state transition probability matrix,

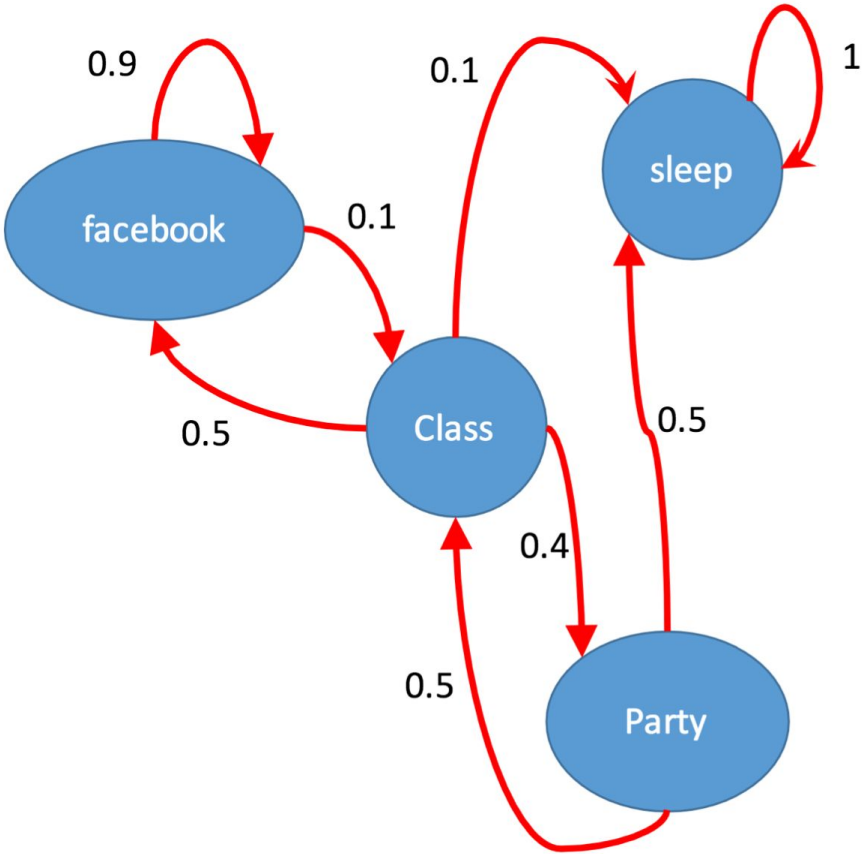
$$P_{s,s'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

There is no action or decision in Markov Processing.

The Markov property is not a limitation on decision process, but on the state.

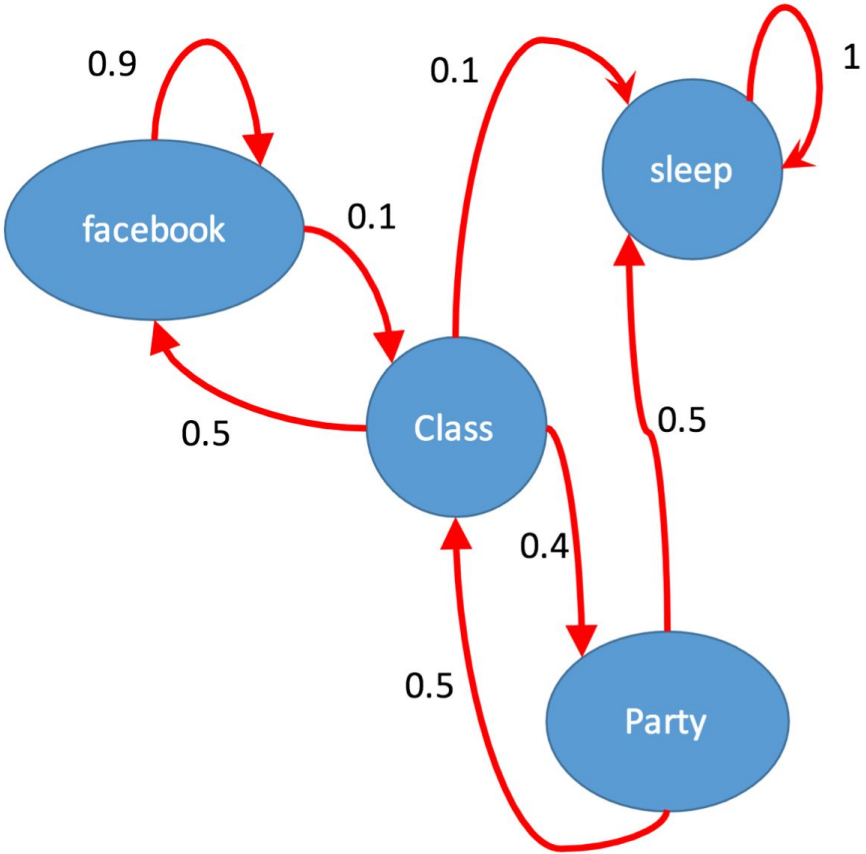
The state must include information about all aspects of the past agent-environment interaction that make a difference for future.

Example: student chain



	FB	C	P	S
FB	0.9	0.1		
C	0.5		0.4	0.1
P		0.5		0.5
S				1

Example: student chain



Sample episodes for Student Markov Chain starting from state Class

- C F F C P S
- C F F F F C P S
- ……

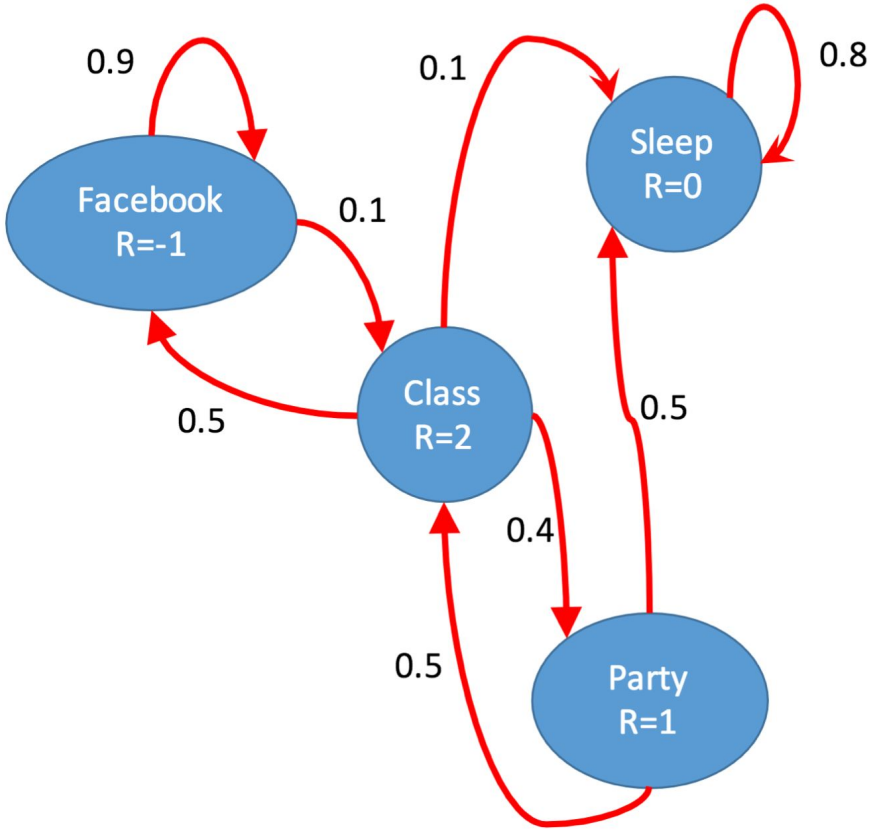
Markov Reward Process

A Markov reward process is a Markov chain with values.

A Markov Reward Process is a tuple $\langle \mathcal{S}, \boldsymbol{P}, \boldsymbol{r}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\boldsymbol{P}$ is a state transition probability matrix,
  $P_{s,s'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
- $\boldsymbol{r}$ is a reward function $r(s) = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is a discount factor $\gamma \in [0, 1]$

Example: student reward chain



	FB	C	P	S
FB	0.9	0.1		
C	0.5		0.4	0.1
P		0.5		0.5
S				1

Goals and rewards

Is a scalar reward signal an adequate notion of a goal? —maybe not, but it is surprisingly flexible.

A goal should specify what we want to achieve, not how we want to achieve it.

A goal must be outside the agent's direct control—thus outside the agent.

The agent must be able to measure success:

- Explicitly
- Frequently during its lifespan.

The reward hypothesis

Agent goal: **maximize the total amount of reward it receives. This means maximizing not immediate reward, but cumulative reward in the long run.**

Reward Hypothesis

> That all of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (called reward)

Example:

- Chess: +/- for winning/losing the game
- Humanoid robot walk: + for forward motion, - for falling over

Return

Suppose the sequence of rewards after step t is:

$$R_{t+1}, R_{t+2}, R_{t+3}, \ldots$$

What do we want to maximize?

At least three cases, but in all of them we seek to maximize the the expected return $\mathbb{E}[G_t]$ on step t

Total reward, G_t = sum of all future reward in the episode
Discounted reward, G_t = sum of all future discounted reward
Average reward, G_t = average reward per time step

From Reward to Return

The objective in RL is to maximize long-term future reward

That is, to choose A_t so as to maximize $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$

But what exactly should be maximized?

The discounted return at time t:

Definition

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Discount Factor $\gamma \in [0, 1]$

$\gamma = 0$ Only care about immediate reward

$\gamma = 1$ Future reward is as beneficial as immediate reward

Episodic tasks vs Continuing tasks

Episodic tasks

- finite number of time steps
- such a sequence is called an *episode*

Continuing tasks

- infinite number of time steps

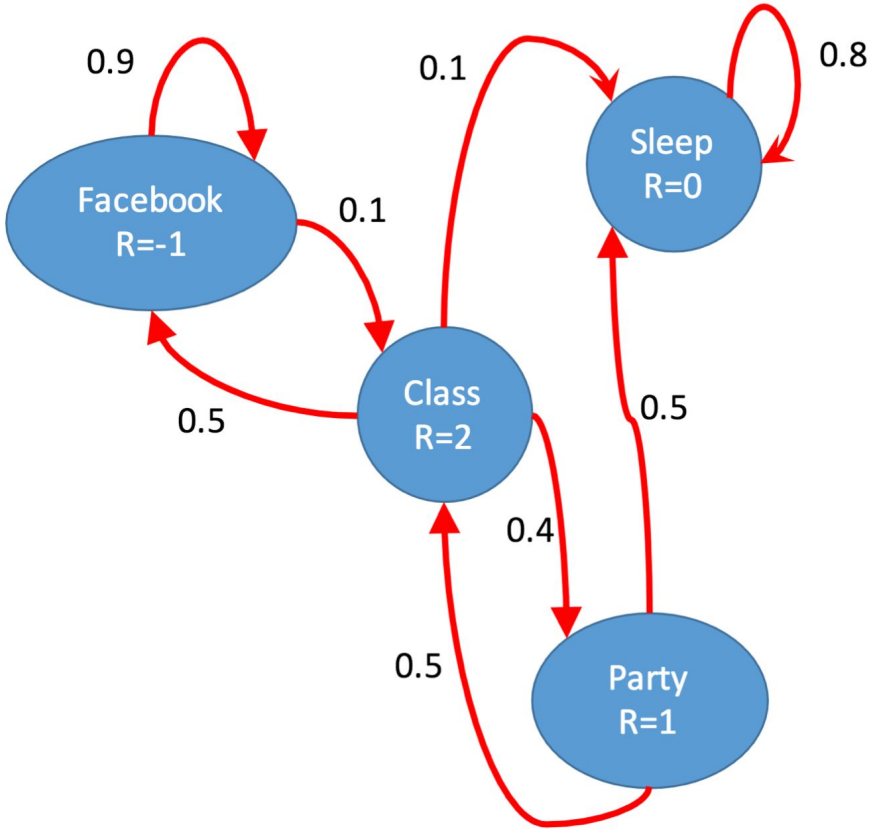
Why discounted?

Most Markov reward and decision processes are discounted. Why?

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \qquad 0 \leq \gamma < 1$$

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behavior shows preference for immediate reward
- It is sometimes possible to use $\gamma = 1$, e.g. if all sequences terminate.

Example: student reward chain



Example of episode:

C F F S

Discount factor γ = 0.5
Total reward:

$G_1 = 2 + 0.5 \times (-1) + (0.5)^2 \times (-1) + (0.5)^3 \times 0$

How about episode
C F F F F C P S?

The value function $v(s)$

The **value function** in a Markov Reward Process is the expected return starting from a state

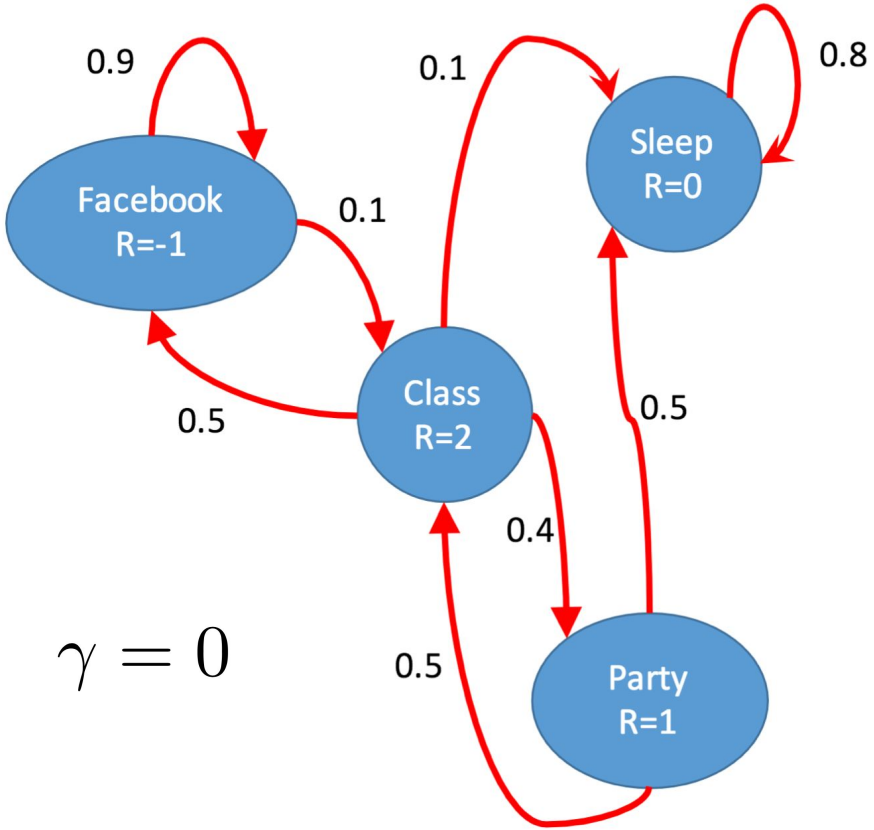
$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

$$G_t = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}$$

Mapping from each state to a real number
More precisely, this is called **state value function**

The value function $v(s)$ gives the long-term value of state s, estimating how good it is for the agent to be in a given state

Value function for student chain



$\gamma = 0$

	FB	C	P	S
FB	0.9	0.1		
C	0.5		0.4	0.1
P		0.5		0.5
S				1

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

$$G_t = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}$$

How to compute value function?

The value function can be decomposed into two parts:

immediate reward + discounted value of successor state

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

How to compute value function?

The value function can be decomposed into two parts:

immediate reward + discounted value of successor state

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

So

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \dots \right) \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma \left(G_{t+1} \right) \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]
\end{aligned}
$$

Bellman Equation

We can write the **Bellman Equation** as:

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s] \\
&= r(s) + \gamma \sum_{s'} P_{s,s'} v(s')
\end{aligned}
$$

Bellman Equation in matrix form

We can write the **Bellman Equation** as:

$$\begin{aligned} v(s) &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s] \\ &= r(s) + \gamma \sum_{s'} P_{s,s'} v(s') \end{aligned}$$

The Bellman equation can be expressed concisely using matrices,

$$\boldsymbol{v} = \boldsymbol{r} + \gamma \boldsymbol{P} \boldsymbol{v}$$

where $\boldsymbol{r} \in \mathbb{R}^n$ and $\boldsymbol{v} \in \mathbb{R}^n$ are column vectors with one entry per state where n is the number states

$$\begin{bmatrix} v(1) \\ v(2) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ \vdots \\ r(n) \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \vdots & \vdots & \dots & \vdots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ v(2) \\ \vdots \\ v(n) \end{bmatrix}$$

Bellman Equation in matrix form

The Bellman equation is a linear system.
It can be solved directly:

$$\boldsymbol{v} = \boldsymbol{r} + \gamma \boldsymbol{P} \boldsymbol{v}$$

$$(\boldsymbol{I} - \gamma \boldsymbol{P})\, \boldsymbol{v} = \boldsymbol{r}$$

$$\boldsymbol{v} = (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1}\, \boldsymbol{r}$$

Computational complexity is $O(n^3)$ for n states (for computing the inverse).

Note: This inverse always exists for $0 \leq \gamma < 1$

Markov Decision Process

A **Markov Decision Process** (**MDP**) is a Markov Reward Process with decisions.

It is an environment in which all states are Markov.

MDP formally describes an environment for Reinforcement Learning

If a reinforcement learning task has the Markov Property, it is basically a Markov Decision Process (MDP).

Markov Decision Process

A Markov decision process (MDP) is a Markov Reward Process with decisions.

If state, action, reward sets are finite, it is a finite MDP.

A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$

$\mathcal{S}$ is a finite set of states

$\mathcal{A}$ is a finite set of actions

p is a state transition probability matrix for each action a

$$p(s, a, s') = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$$

r is a reward function $r(s, a) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$

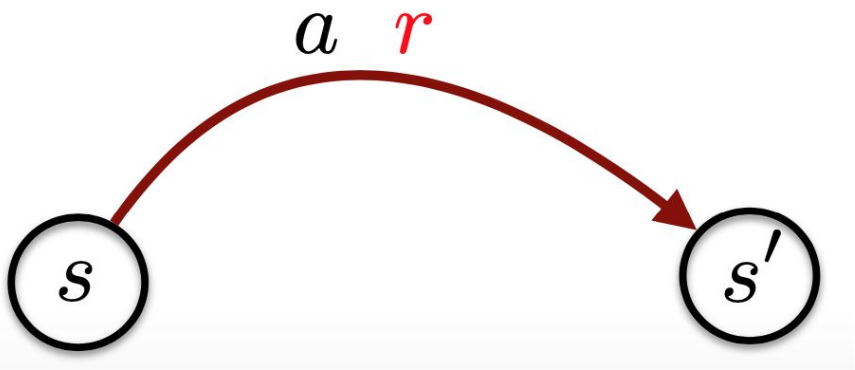
γ is a discount factor $\gamma \in [0, 1]$

To define a finite MDP, you need to give:

- state set, action set, discount factor
- one-step "dynamics"

One step dynamics

One-step dynamics:



$$p(s', r|s, a) = \mathbb{P}\left[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\right]$$

This function defines the dynamics of the MDP.

It completely characterizes the environment dynamics.

That is, the probability of each possible value for S_{t+1} and R_{t+1} depends only on the immediately preceding state S_t and action A_t

We can compute anything we want to know about the environment from p

State transition probabilities

From the p , we can compute the state transition probabilities:

$$p(s'|s,a) = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right] = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

Expected Reward

We can also compute the expected reward for state-action pairs:

$$
\begin{aligned}
r(s, a) &= \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a\right] \\
&= \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)
\end{aligned}
$$

An example of finite MDP

Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: high, low.

State Set?
Action Set?
One-Step Dynamics?

Recycling robot MDP

$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$

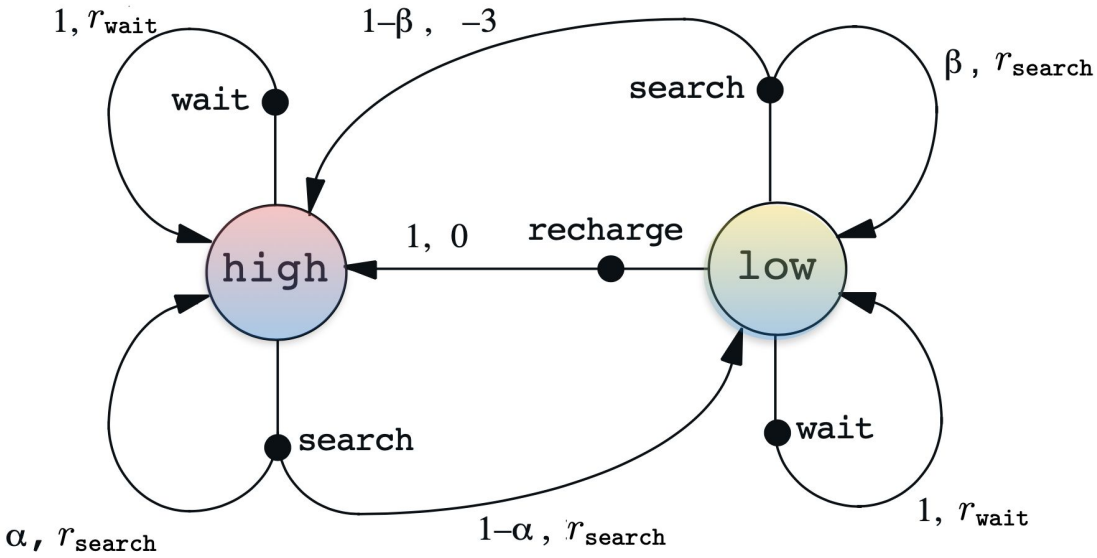
$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$r_{\texttt{search}} > r_{\texttt{wait}}$

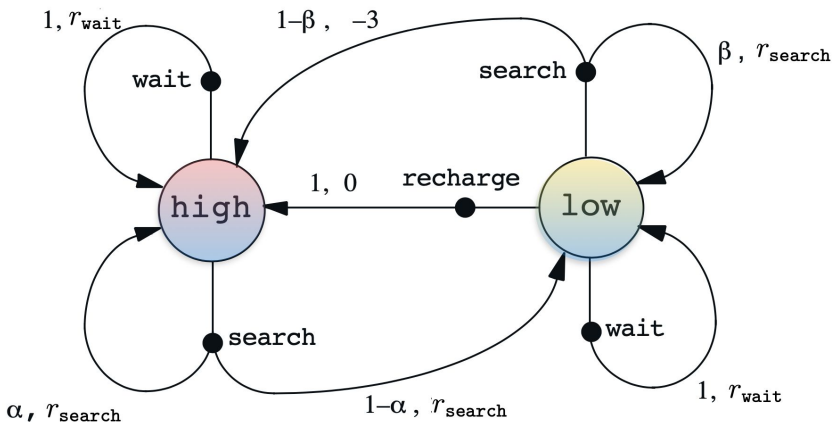


Recycling robot MDP

$$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$$

$$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$$

$$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$$



s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1 - \alpha$	$r_{\texttt{search}}$
low	search	high	$1 - \beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	$r_{\texttt{wait}}$
low	wait	high	0	$r_{\texttt{wait}}$
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	0

Policy

A policy is a mapping from states to probabilities of selecting each possible action

A policy π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a | S_t = s\right]$$

The agent learns a policy in RL

A policy is a mapping from states to probabilities of selecting each possible action

A policy π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a | S_t = s\right]$$

If the agent is following policy π at time t, then $\pi(a \mid s)$ is the probability that $A_t = a$ if $S_t = s$

A policy fully defines the behavior of an agent

Reinforcement learning methods specify how the agent changes its policy as a result of experience.

Special case - deterministic policies:

$\pi_t(s)$ = the action taken with prob=1 when $S_t = s$

The relationship between MDP and MRP

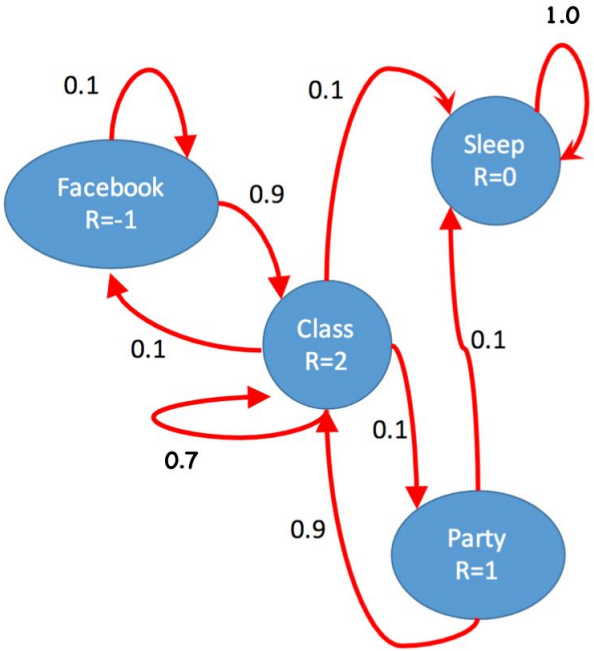
An interesting aspect of specifying a stationary policy π on a MDP is that evaluating the value function for the policy is equivalent to evaluating the value function on an equivalent Markov reward process. Specifically we define the Markov reward process $M'(S, \mathbf{P}^\pi, R^\pi, \gamma)$, where $\mathbf{P}^\pi$ and R^π are given by:

$$R^\pi(s) = \sum_{a \in A} \pi(a|s) R(s,a) \ ,$$
$$P^\pi(s'|s) = \sum_{a \in A} \pi(a|s) P(s'|s,a) \ .$$

http://web.stanford.edu/class/cs234/CS234Win2019/slides/lnotes2.pdf

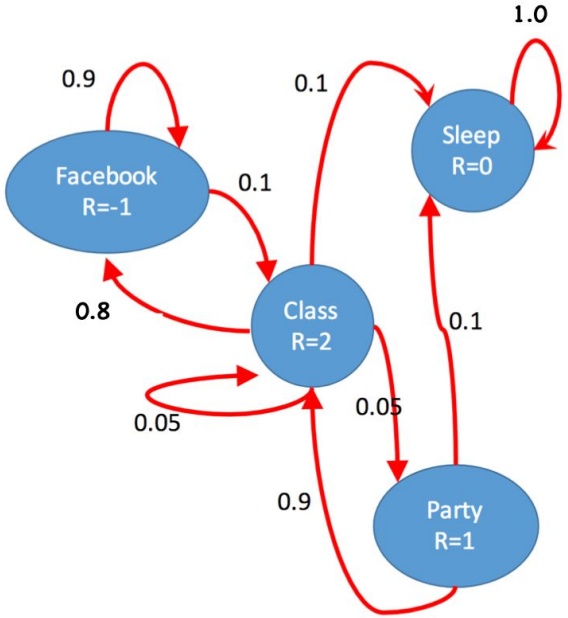
Example

Study



π_{study}

Facebook



π_{facebook}

Two types of Value functions: $v_\pi(s)$ and $q_\pi(s, a)$

1. (**state value function**) A value functions estimates how good it is for the agent to be in a given state

2. (**state-action value function**, or **action value function**) A value functions estimates how good it is for the agent to perform a given action in a given state

The notion of “how good” is defined in terms of future rewards that can be expected (i.e., expected return)

Almost all RL algorithms involve estimating value functions (functions of states or of state-actions pairs)

State Value function $v_\pi(s)$

The value of a state for a policy is the expected return starting from that state; it depends on the agent's policy:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]
\end{aligned}
$$

The expected return starting from state s, and then following policy π

The value function $v_\pi(s)$ gives the long-term value of state s under the policy π

v_π is called the *state-value function* or *value function* for policy π

Bellman Equation for $v_\pi(s)$

The value function can be decomposed into two parts:

immediate reward + discounted value of successor state

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

So:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} | S_t = s \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \right]
\end{aligned}
$$

Bellman Equation for $v_\pi(s)$

A key property of value-functions is their recursive relationships:

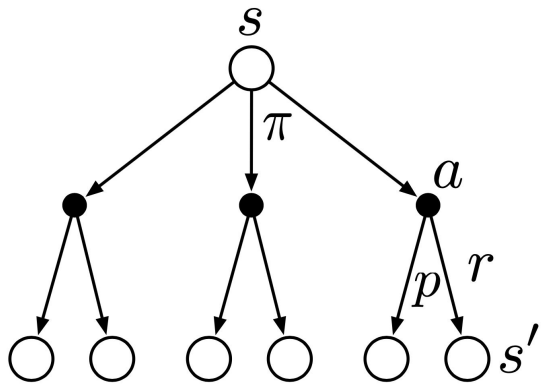
$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \\
&= \sum_a \pi(a \mid s)\left[\sum_{s'}\sum_r p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right]\right] \\
&= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]
\end{aligned}
$$

Bellman Equation and Backup Diagram for $v_\pi(s)$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big]$$

This is a set of equations (in fact, linear), one for each state. The value function for is its unique solution.

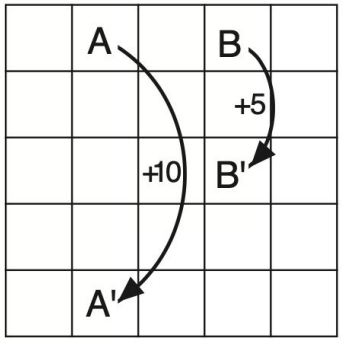
Backup diagram

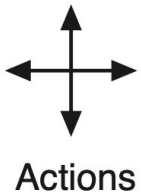


Backup diagram for v_π

Gridworld

Example 3.5: Gridworld Figure 3.2 (left) shows a rectangular gridworld representation of a simple finite MDP. The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: `north`, `south`, `east`, and `west`, which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of $+10$ and take the agent to A′. From state B, all actions yield a reward of $+5$ and take the agent to B′.

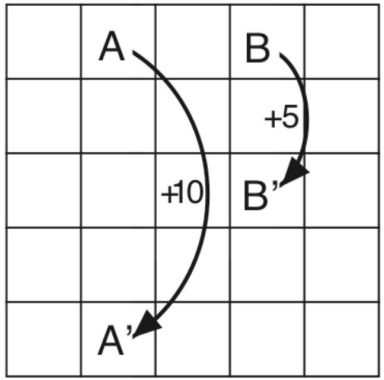




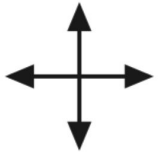
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

State-value function for equiprobable random policy; $\gamma = 0.9$

Gridworld



(a)



Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

SOME OBSERVATIONS:

- The values are obtained by solving linear system from Bellman equation
- Notice the negative values near the lower edge!
- State A is the best state to be in but its expected return is less than 10 (immediate reward)
- State B is valued more than 5 (immediate reward)
- The Bellman equations holds for all states. Check!

State-Action Value function $q_\pi(s, a)$

The value of an action (in a state) is the expected return starting after taking that action from that state; depends on the agent's policy:

$$
\begin{aligned}
q_\pi(s, a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\
&= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]
\end{aligned}
$$

The expected return starting from state S, taking action a, and then following policy π (Note that this action a can be any action, not necessarily following π)

$q_\pi(s, a)$ is called the *state-action value function* or *action value function* for π

Bellman Equation for $q_\pi(s, a)$

The value function can be decomposed into two parts:

immediate reward + discounted value of successor state

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

So for state value function:

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

Similarly, for state-action value function

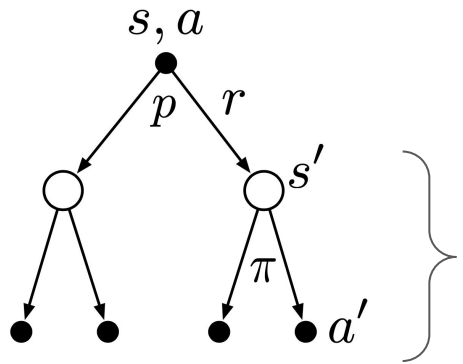
$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right]$$

Bellman Equation and Backup Diagram for $q_\pi(s, a)$

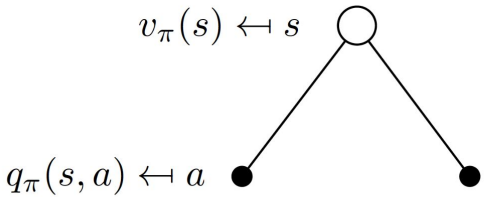
$$
\begin{aligned}
q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \\
&= \sum_{s', r} p(s', r | s, a)[r + \gamma v_\pi(s')] \\
&= \sum_{s', r} p(s', r | s, a)[r + \gamma \sum_{a'} \pi(a' | s') q_\pi(s', a')]
\end{aligned}
$$

Bellman Equation and Backup Diagram for $q_\pi(s, a)$

$$
\begin{aligned}
q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \\
&= \sum_{s', r} p(s', r | s, a)[r + \gamma v_\pi(s')] \\
&= \sum_{s', r} p(s', r | s, a)[r + \gamma \sum_{a'} \pi(a' | s') q_\pi(s', a')]
\end{aligned}
$$



q_π backup diagram



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

Optimal Value Function and Optimal Policy

- For finite MDPs, policies can be partially ordered:

$$\pi \geq \pi' \quad \text{if and only if} \quad v_\pi(s) \geq v_{\pi'}(s) \ \text{ for all } \ s \in \mathcal{S}$$

Optimal Value Function and Optimal Policy

- For finite MDPs, policies can be partially ordered:

$$\pi \geq \pi' \quad \text{if and only if} \quad v_\pi(s) \geq v_{\pi'}(s) \ \text{ for all } \ s \in \mathcal{S}$$

- There are always one or more policies that are better than or equal to all the others. These are the optimal policies. We denote them all π_*

Optimal Value Function and Optimal Policy

- For finite MDPs, policies can be partially ordered:

$$\pi \geq \pi' \quad \text{if and only if} \quad v_\pi(s) \geq v_{\pi'}(s) \ \text{ for all } \ s \in \mathcal{S}$$

- There are always one or more policies that are better than or equal to all the others. These are the optimal policies. We denote them all π_*
- Optimal policies share the same optimal state-value function:

$$v_*(s) = \max_\pi v_\pi(s) \quad \text{for all } s \in \mathcal{S}$$

- Optimal policies also share the same optimal action-value function:

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}$$

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

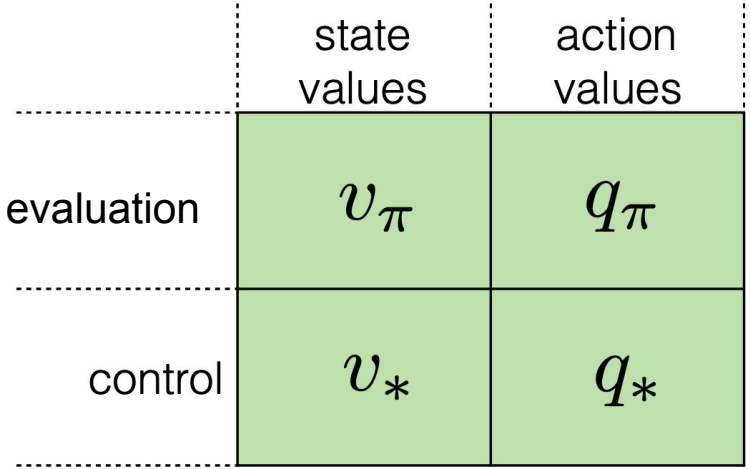
Optimal Value Function and Optimal Policy

The optimal value functions is unique, but there can be multiple optimal policies!

To search for optimal Policy, it is sufficient just to look for deterministic policies.

In a finite MDP, the number of deterministic policies is finite, and equals $|A|^{|S|}$

4 value functions



All theoretical objects, mathematical ideals (expected values)

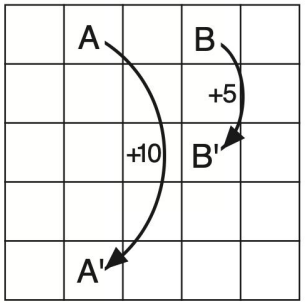
From Optimal Value Function to Optimal Policy

Any policy that is greedy with respect to v_* is an optimal policy.

Therefore, given v_*, one-step-ahead search produces the long-term optimal actions.

$$\pi_*(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

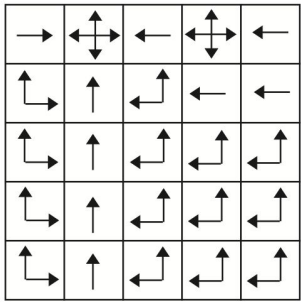
E.g., back to the gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

From Optimal Value Function to Optimal Policy

Given q_*, the agent does not even have to do a one-step-ahead search:

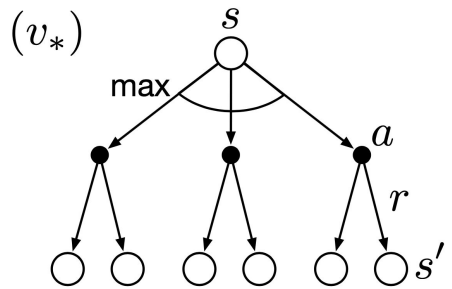
$$\pi_*(s) = \arg \max_{a \in \mathcal{A}(s)} q_*(s, a)$$

Bellman Optimality Equation and Backup Diagram for v_*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} v_*(s) &= \max_a q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big]. \end{aligned}$$

Backup diagram

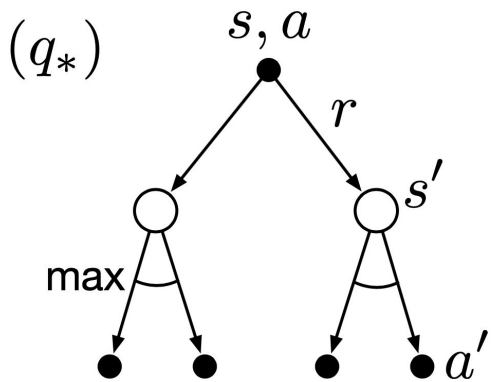


v_* is the unique solution of this system of nonlinear equations

Bellman Optimality Equation and Backup Diagram for q_*

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right].
\end{aligned}
$$

Backup diagram



q_* is the unique solution of this system of nonlinear equations

Solving the Bellman optimality equation

Finding an optimal policy by solving the Bellman Optimality Equation requires the following:

- Accurate knowledge of environment dynamics
- we have enough space and time to do the computation
- the Markov Property

How much space and time do we need?

- Polynomial in number of states (via dynamic programming methods; Chapter 4)
- BUT, number of states is often huge (e.g., backgammon has about 10^{20} states).

We usually have to settle for approximations.

Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

Summary

Agent-environment interaction

- **States**
- **Actions**
- **Rewards**

Policy: stochastic rule for selecting actions

Return: the function of future rewards agent tries to maximize

Episodic and **continuing** tasks

Markov Property and **Markov Decision Process**

- Transition probabilities
- Expected rewards

Value functions

- **State-value function** for a policy
- **Action-value function** for a policy

Optimal value function
Optimal policy
Bellman Equation
Bellman Optimality Equation

The need for approximation to solve Bellman Optimality Equation