Lecture 3:
Markov Decision Processes and Dynamic Programming

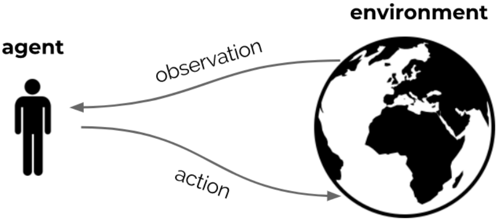
Diana Borsa

January 15, 2021

Background

Sutton & Barto 2018, Chapter 3 + 4

Recap



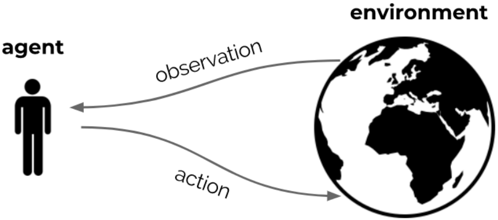
- Reinforcement learning is the science of learning to make decisions
- Agents can learn a policy, value function and/or a model
- The general problem involves taking into account time and consequences
- Decisions affect the reward, the agent state, and environment state

This Lecture

- Last lecture: multiple actions, but only one state—no model
- This lecture:
  - Formalise the problem with full sequential structure
  - Discuss first class of solution methods which assume true model is given
  - These methods are called dynamic programming
- Next lectures: use similar ideas, but use sampling instead of true model

Formalising the RL interaction

Formalising the RL interface



- We will discuss a mathematical formulation of the agent-environment interaction
- This is called a Markov Decision Process (MDP)
- Enables us to talk clearly about the objective and how to achieve it

MDPs: A simplifying assumption

- For now, assume the environment is fully observable:
  $\Rightarrow$ the current observation contains all relevant information
- Note: Almost all RL problems can be formalised as MDPs, e.g.,
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs
  - Bandits are MDPs with one state

Markov Decision Process

Definition (Markov Decision Process - *Sutton & Barto 2018*)

A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where

- $\mathcal{S}$ is the set of all possible states
- $\mathcal{A}$ is the set of all possible actions (e.g., motor controls)
- $p(r, s' \mid s, a)$ is the joint probability of a reward r and next state s', given a state s and action a
- $\gamma \in [0, 1]$ is a discount factor that trades off later rewards to earlier ones

Observations:

- p defines the dynamics of the problem
- Sometimes it is useful to marginalise out the state transitions or expected reward:

$$p(s' \mid s, a) = \sum_r p(s', r \mid s, a) \qquad \mathbb{E}[R \mid s, a] = \sum_r r \sum_{s'} p(r, s' \mid s, a) .$$

Markov Decision Process: Alternative Definition

Definition (Markov Decision Process)

A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where

- $\mathcal{S}$ is the set of all possible states
- $\mathcal{A}$ is the set of all possible actions (e.g., motor controls)
- $p(s' \mid s, a)$ is the probability of transitioning to s', given a state s and action a
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the excepted reward, achieved on a transition starting in (s, a)

$$r = \mathbb{E}\left[R \mid s, a\right]$$

- $\gamma \in [0, 1]$ is a discount factor that trades off later rewards to earlier ones

Note: These are equivalent formulations: no additional assumptions w.r.t the previous def.

Markov Property: *The future is independent of the past given the present*

Definition (Markov Property)

Consider a sequence of random variables, $\{S_t\}_{t \in \mathbb{N}}$, indexed by time. A state s has the Markov property when for states $\forall s' \in \mathcal{S}$

$$p\left(S_{t+1} = s' \mid S_t = s\right) = p\left(S_{t+1} = s' \mid h_{t-1}, S_t = s\right)$$

for all possible histories $h_{t-1} = \{S_1, \ldots, S_{t-1}, A_1, \ldots, A_{t-1}, R_1, \ldots, R_{t-1}\}$

In a Markov Decision Process all states are assumed to have the Markov property.

- The state captures all relevant information from the history.
- Once the state is known, the history may be thrown away.
- The state is a sufficient statistic of the past.

Markov Property in a MDP: Test your understanding

In a Markov Decision Process all states are assumed to have the Markov property.

Q: In an MDP this property implies: (Which of the following statements are true?)

$$p\left(S_{t+1}=s' \mid S_t=s, A_t=a\right)=p\left(S_{t+1}=s' \mid S_1,\ldots,S_{t-1},A_1,\ldots,A_t,S_t=s\right) \quad (1)$$

$$p\left(S_{t+1}=s' \mid S_t=s, A_t=a\right)=p\left(S_{t+1}=s' \mid S_1,\ldots,S_{t-1},S_t=s,A_t=a\right) \quad (2)$$

$$p\left(S_{t+1}=s' \mid S_t=s, A_t=a\right)=p\left(S_{t+1}=s' \mid S_1,\ldots,S_{t-1},S_t=s\right) \quad (3)$$

$$p\left(R_{t+1}=r, S_{t+1}=s' \mid S_t=s\right)=p\left(R_{t+1}=r, S_{t+1}=s' \mid S_1,\ldots,S_{t-1},S_t=s\right) \quad (4)$$

Example: cleaning robot

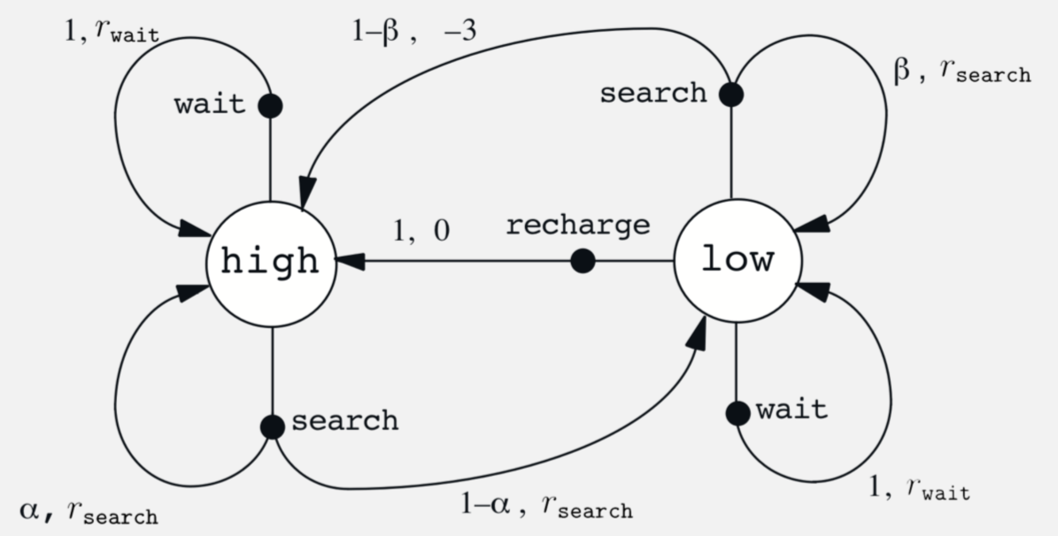
- Consider a robot that cleans soda cans
- Two states: high battery charge or low battery charge
- Actions: {wait, search} in high, {wait, search, recharge} in low
- Dynamics may be stochastic
  - $p(S_{t+1} = \text{high} \mid S_t = \text{high}, A_t = \text{search}) = \alpha$
  - $p(S_{t+1} = \text{low} \mid S_t = \text{high}, A_t = \text{search}) = 1 - \alpha$
- Reward could be expected number of collected cans (deterministic), or actual number of collected cans (stochastic)

Reference: Sutton and Barto, Chapter 3, pg 52-53.

Example: robot MDP

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

Example: robot MDP



Formalising the objective

Returns

- Acting in a MDP results in immediate rewards R_t, which leads to returns G_t:
  - Undiscounted return (episodic/finite horizon pb.)

$$G_t = R_{t+1} + R_{t+2} + \dots + R_T = \sum_{k=0}^{T-t-1} R_{t+k+1}$$

  - Discounted return (finite or infinite horizon pb.)

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

  - Average return (continuing, infinite horizon pb.)

$$G_t = \frac{1}{T-t-1}\left(R_{t+1} + R_{t+2} + \dots + R_T\right) = \frac{1}{T-t-1} \sum_{k=0}^{T-t-1} R_{t+k+1}$$

Note: These are random variables that depends on MDP and policy

Discounted Return

- Discounted returns G_t for infinite horizon $T \to \infty$:

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

- The discount $\gamma \in [0, 1]$ is the present value of future rewards
  - The marginal value of receiving reward R after $k+1$ time-steps is $\gamma^k R$
  - For $\gamma < 1$, immediate rewards are more important than delayed rewards
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Why discount?

Most Markov decision processes are discounted. Why?

- Problem specification:
  - Immediate rewards may actually be more valuable (e.g., consider earning interest)
  - Animal/human behaviour shows preference for immediate reward
- Solution side:
  - Mathematically convenient to discount rewards
  - Avoids infinite returns in cyclic Markov processes
- The way to think about it: reward and discount together determine the goal

Policies

Goal of an RL agent

To find a behaviour policy that maximises the (expected) return G_t

- A policy is a mapping $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that, for every state s assigns for each action $a \in \mathcal{A}$ the probability of taking that action in state s. Denoted by $\pi(a|s)$.
- For deterministic policies, we sometimes use the notation $a_t = \pi(s_t)$ to denote the action taken by the policy.

Value Functions

- The value function $v(s)$ gives the long-term value of state s

$$v_\pi(s) = \mathbb{E}\left[G_t \mid S_t = s, \pi\right]$$

- We can define (state-)action values:

$$q_\pi(s, a) = \mathbb{E}\left[G_t \mid S_t = s, A_t = a, \pi\right]$$

- (Connection between them) Note that:

$$v_\pi(s) \;=\; \sum_a \pi(a \mid s) q_\pi(s, a) \;=\; \mathbb{E}\left[q_\pi(S_t, A_t) \mid S_t = s, \pi\right] \;\;, \forall s$$

Optimal Value Function

Definition (Optimal value functions)

The optimal state-value function $v^*(s)$ is the maximum value function over all policies

$$v^*(s) = \max_\pi v_\pi(s)$$

The optimal action-value function $q^*(s, a)$ is the maximum action-value function over all policies

$$q^*(s, a) = \max_\pi q_\pi(s, a)$$

- The optimal value function specifies the best possible performance in the MDP
- An MDP is "solved" when we know the optimal value function

Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \ , \ \forall s$$

Theorem (Optimal Policies)

For any Markov decision process

- *There exists an optimal policy* π^* *that is better than or equal to all other policies,* $\pi^* \geq \pi, \forall \pi$
  (There can be more than one such optimal policy.)
- *All optimal policies achieve the optimal value function,* $v^{\pi^*}(s) = v^*(s)$
- *All optimal policies achieve the optimal action-value function,* $q^{\pi^*}(s,a) = q^*(s,a)$

Finding an Optimal Policy

An optimal policy can be found by maximising over $q^*(s, a)$,

$$\pi^*(s, a) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q^*(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

Observations:

- There is always a deterministic optimal policy for any MDP
- If we know $q^*(s, a)$, we immediately have the optimal policy
- There can be multiple optimal policies
- If multiple actions maximize $q_*(s, \cdot)$, we can also just pick any of these (including stochastically)

Bellman Equations

Value Function

- The value function $v(s)$ gives the long-term value of state s

$$v_\pi(s) = \mathbb{E}\left[G_t \mid S_t = s, \pi\right]$$

- It can be defined recursively:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s, \pi\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t \sim \pi(S_t)\right] \\
&= \sum_a \pi(a \mid s) \sum_r \sum_{s'} p(r, s' \mid s, a)\left(r + \gamma v_\pi(s')\right)
\end{aligned}$$

- The final step writes out the expectation explicitly

Action values

- We can define state-action values

$$q_\pi(s, a) = \mathbb{E}\left[G_t \mid S_t = s, A_t = a, \pi\right]$$

- This implies

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

- Note that

$$v_\pi(s) \;=\; \sum_a \pi(a \mid s) q_\pi(s, a) \;=\; \mathbb{E}\left[q_\pi(S_t, A_t) \mid S_t = s, \pi\right] \;, \; \forall s$$

Bellman Equations

Theorem (Bellman Expectation Equations)

Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, for any policy π, the value functions obey the following expectation equations:

$$v_\pi(s) = \sum_a \pi(s,a) \left[r(s,a) + \gamma \sum_{s'} p(s'|a,s) v_\pi(s') \right] \tag{5}$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|a,s) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a') \tag{6}$$

The Bellman Optimality Equations

Theorem (Bellman Optimality Equations)

Given an MDP, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, the optimal value functions obey the following expectation equations:

$$v^*(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} p(s'|a,s) v^*(s') \right] \tag{7}$$

$$q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|a,s) \max_{a' \in \mathcal{A}} q^*(s',a') \tag{8}$$

There can be no policy with a higher value than $v_*(s) = \max_\pi v_\pi(s)$, $\forall s$

Some intuition

(Reminder) Greedy on v^* = Optimal Policy

- An optimal policy can be found by maximising over $q^*(s, a)$,

$$\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\, q^*(s,a) \\ 0 & otherwise \end{cases}$$

- Apply the Bellman Expectation Eq. (6):

$$\begin{aligned} q_{\pi^*}(s,a) &= r(s,a) + \gamma \sum_{s'} p(s'|a,s) \underbrace{\sum_{a' \in \mathcal{A}} \pi^*(a'|s') q_{\pi^*}(s',a')}_{max_{a'} q^*(s',a')} \\ &= r(s,a) + \gamma \sum_{s'} p(s'|a,s) \max_{a' \in \mathcal{A}} q^*(s',a') \end{aligned}$$

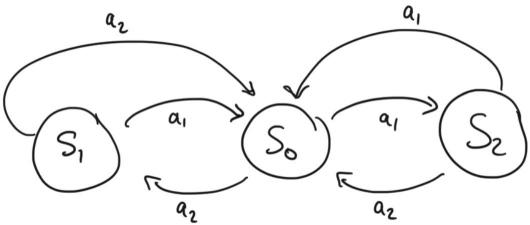
Solving RL problems using the Bellman Equations

Problems in RL

- *Pb1:* Estimating v_π or q_π is called policy evaluation or, simply, prediction
  - Given a policy, what is my expected return under that behaviour?
  - Given this treatment protocol/trading strategy, what is my expected return?
- *Pb2*: Estimating v_* or q_* is sometimes called control, because these can be used for policy optimisation
  - What is the optimal way of behaving? What is the optimal value function?
  - What is the optimal treatment? What is the optimal control policy to minimise time, fuel consumption, etc?

Exercise:

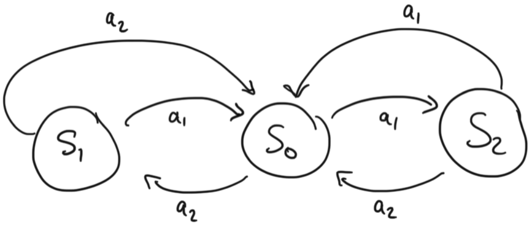
- Consider the following MDP:



  - The actions have a 0.9 probability of success and with 0.1 probably we remain in the same state
  - $R_t = 0$ for all transitions that end up in S_0, and $R_t = -1$ for all other transitions

Exercise: (pause to work this out)

- Consider the following MDP:



  - The actions have a 0.9 probability of success and with 0.1 probably we remain in the same state
  - $R_t = 0$ for all transitions that end up in S_0, and $R_t = -1$ for all other transitions

- **Q:** Evaluation problems (Consider a discount $\gamma = 0.9$)
  - What is v_π for $\pi(s) = a_1(\rightarrow), \forall s$?
  - What is v_π for the uniformly random policy?
  - Same policy evaluation problems for $\gamma = 0.0$? (What do you notice?)

A solution

Bellman Equation in Matrix Form

- The Bellman value equation, for given π, can be expressed using matrices,

$$\mathbf{v} = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}$$

where

$$\begin{aligned} v_i &= v(s_i) \\ r_i^\pi &= \mathbb{E}\left[R_{t+1} \mid S_t = s_i, A_t \sim \pi(S_t)\right] \\ P_{ij}^\pi &= p(s_j \mid s_i) = \sum_a \pi(a \mid s_i) p(s_j \mid s_i, a) \end{aligned}$$

Bellman Equation in Matrix Form

- The Bellman equation, for a given policy π, can be expressed using matrices,

$$\mathbf{v} = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}$$

- This is a linear equation that can be solved directly:

$$\begin{aligned} \mathbf{v} &= \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{v} \\ (\mathbf{I} - \gamma \mathbf{P}^\pi)\, \mathbf{v} &= \mathbf{r}^\pi \\ \mathbf{v} &= (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1}\, \mathbf{r}^\pi \end{aligned}$$

- Computational complexity is $O(|\mathcal{S}|^3)$ — only possible for small problems
- There are iterative methods for larger problems
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Solving the Bellman Optimality Equation

- The Bellman optimality equation is non-linear
- Cannot use the same direct matrix solution as for policy optimisation (in general)

- Many iterative solution methods:
  - Using models / dynamic programming
    - Value iteration
    - Policy iteration
  - Using samples
    - Monte Carlo
    - Q-learning
    - Sarsa

Dynamic Programming

Dynamic Programming

The 1950s were not good years for mathematical research. I felt I had to shield the Air Force from the fact that I was really doing mathematics. What title, what name, could I choose? I was interested in planning, in decision making, in thinking. But planning is not a good word for various reasons. I decided to use the word 'programming.' I wanted to get across the idea that this was dynamic, this was time-varying—I thought, let's kill two birds with one stone. Let's take a word that has a precise meaning, namely dynamic, in the classical physical sense. It also is impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.

– Richard Bellman
(slightly paraphrased for conciseness)

Dynamic programming

> *Dynamic programming refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP).*

Sutton & Barto 2018

- We will discuss several dynamic programming methods to solve MDPs
- All such methods consist of two important parts:

policy evaluation and policy improvement

Policy evaluation

- We start by discussing how to estimate

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

- Idea: turn this equality into an update

Algorithm

- First, initialise v_0, e.g., to zero
- Then, iterate

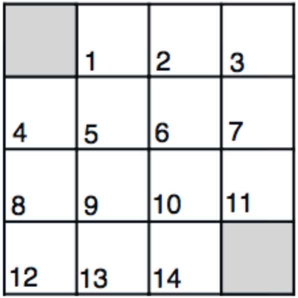
$$\forall s: \quad v_{k+1}(s) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid s, \pi\right]$$

- Stopping: whenever $v_{k+1}(s) = v_k(s)$, for all s, we must have found v_π

- *Q*: Does this algorithm always converge?
  Answer: Yes, under appropriate conditions (e.g., $\gamma < 1$). More next lecture!

Example: Policy evaluation

actions



$$R_t = -1$$

on all transitions

Policy evaluation

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

Policy evaluation

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

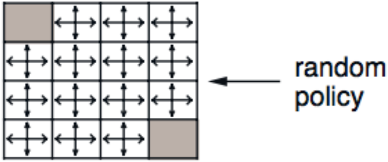
$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy evaluation + Greedy Improvement

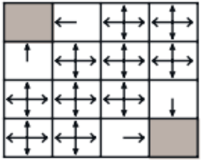
$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



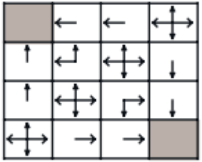
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Policy evaluation + Greedy Improvement

$k = 3$

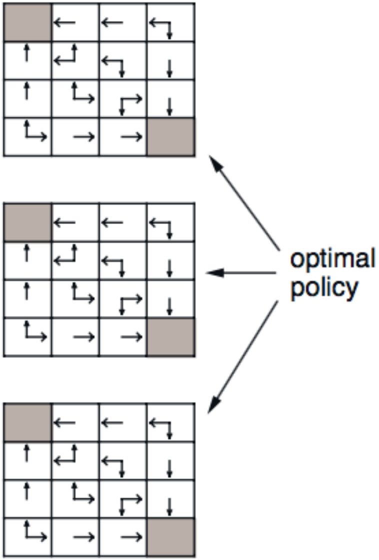
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Improvement

- The example already shows we can use evaluation to then improve our policy
- In fact, just being greedy with respect to the values of the random policy sufficed! (That is not true in general)

Algorithm

Iterate, using

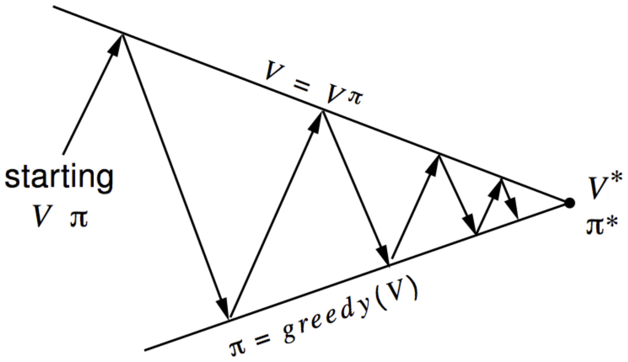
$$\forall s: \quad \pi_{\text{new}}(s) = \operatorname*{argmax}_a q_\pi(s, a)$$
$$= \operatorname*{argmax}_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a\right]$$

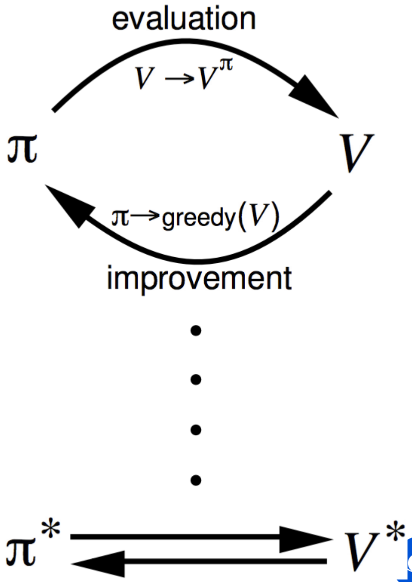
Then, evaluate π_{new} and repeat

- Claim: One can show that $v_{\pi_{\text{new}}}(s) \geq v_\pi(s)$, for all s

Policy Improvement: $q_{\pi_{\text{new}}}(s, a) \geq q_{\pi}(s, a)$

Policy Iteration





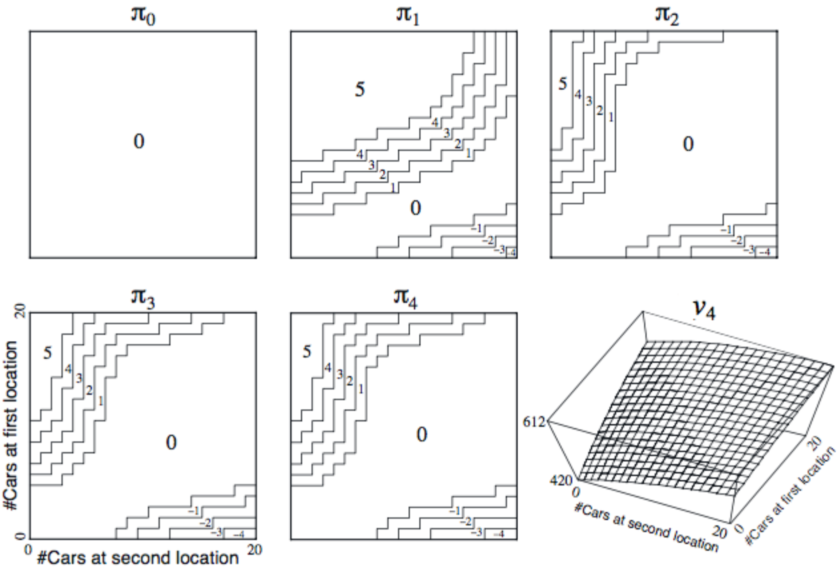
Policy evaluation Estimate v^{π}

Policy improvement Generate $\pi' \geq \pi$

Example: Jack's Car Rental

- States: Two locations, maximum of 20 cars at each
- Actions: Move up to 5 cars overnight (-$2 each)
- Reward: $10 for each available car rented, $\gamma = 0.9$
- Transitions: Cars returned and requested randomly
  - Poisson distribution, n returns/requests with prob $\frac{\lambda^n}{n!} e^{-\lambda}$
  - 1st location: average requests = 3, average returns = 3
  - 2nd location: average requests = 4, average returns = 2

Example: Jack's Car Rental – Policy Iteration



Policy Iteration

- Does policy evaluation need to converge to v^π?
- Or should we stop when we are 'close'?
  (E.g., with a threshold on the change to the values)
  - Or simply stop after k iterations of iterative policy evaluation?
  - In the small gridworld $k = 3$ was sufficient to achieve optimal policy
- Extreme: Why not update policy every iteration — i.e. stop after $k = 1$?
  - This is equivalent to value iteration

Value Iteration

- We could take the Bellman optimality equation, and turn that into an update

$$\forall s: \quad v_{k+1}(s) \leftarrow \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = s\right]$$

- This is equivalent to policy iteration, with $k = 1$ step of policy evaluation between each two (greedy) policy improvement steps

Algorithm: Value Iteration

- Initialise v_0
- Update: $v_{k+1}(s) \leftarrow \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = s\right]$
- Stopping: whenever $v_{k+1}(s) = v_k(s)$, for all s, we must have found v^*

Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

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

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Synchronous Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + (Greedy) Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Observations:

- Algorithms are based on state-value function $v_\pi(s)$ or $v^*(s)$ $\Rightarrow$ complexity $O(|\mathcal{A}||\mathcal{S}|^2)$ per iteration, for $|\mathcal{A}|$ actions and $|\mathcal{S}|$ states
- Could also apply to action-value function $q_\pi(s,a)$ or $q^*(s,a)$ $\Rightarrow$ complexity $O(|\mathcal{A}|^2|\mathcal{S}|^2)$ per iteration

Extensions to Dynamic Programming

Asynchronous Dynamic Programming

- DP methods described so far used synchronous updates (all states in parallel)
- Asynchronous DP
  - backs up states individually, in any order
  - can significantly reduce computation
  - guaranteed to converge if all states continue to be selected

Asynchronous Dynamic Programming

Three simple ideas for asynchronous dynamic programming:

- In-place dynamic programming
- Prioritised sweeping
- Real-time dynamic programming

In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function

$$\text{for all } s \text{ in } \mathcal{S}: \quad v_{\text{new}}(s) \leftarrow \max_a \mathbb{E}\left[R_{t+1} + \gamma v_{\text{old}}(S_{t+1}) \mid S_t = s\right]$$

$$v_{old} \leftarrow v_{new}$$

- In-place value iteration only stores one copy of value function

$$\text{for all } s \text{ in } \mathcal{S}: \quad v(s) \leftarrow \max_a \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

Prioritised Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_a \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right] - v(s) \right|$$

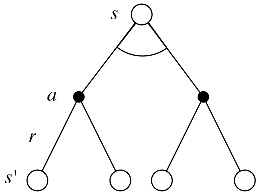
- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

Real-Time Dynamic Programming

- Idea: only update states that are relevant to agent
- E.g., if the agent is in state S_t, update that state value, or states that it expects to be in soon

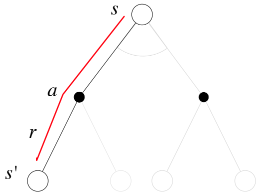
Full-Width Backups

- Standard DP uses full-width backups
- For each backup (sync or async)
  - Every successor state and action is considered
  - Using true model of transitions and reward function
- DP is effective for medium-sized problems (millions of states)
- For large problems DP suffers from curse of dimensionality
  - Number of states $n = |\mathcal{S}|$ grows exponentially with number of state variables
- Even one full backup can be too expensive



Sample Backups

- In subsequent lectures we will consider sample backups
- Using sample rewards and sample transitions $\langle s, a, r, s' \rangle$
  (Instead of reward function r and transition dynamics p)
- Advantages:
  - Model-free: no advance knowledge of MDP required
  - Breaks the curse of dimensionality through sampling
  - Cost of backup is constant, independent of $n = |\mathcal{S}|$



Summary

What have we covered today?

- Markov Decision Processes
- Objectives in an MDP: different notion of return
- Value functions - expected returns, condition on state (and action)
- Optimality principles in MDPs: optimal value functions and optimal policies
- Bellman Equations
- Two class of problems in RL: evaluation and control
- How to compute v_π (aka solve an evaluation/prediction problem)
- How to compute the optimal value function via dynamic programming:
  - Policy Iteration
  - Value Iteration

Questions?

The only stupid question is the one you were afraid to ask but never did.
-Rich Sutton

For questions that may arise during this lecture please use Moodle and/or the next Q&A session.