Lecture 3:
Markov Decision Processes and Dynamic Programming

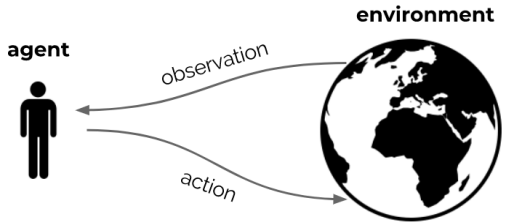
Hado van Hasselt

January 30, 2018, UCL

Background

Sutton & Barto 2018, Chapter 3 + 4

Recap

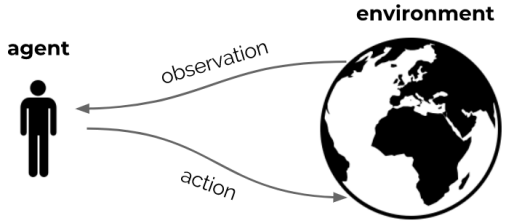


- Reinforcement learning is the science of learning to make decisions
- Agents can learn a policy, value function and/or a model
- The general problem involves taking into account time and consequences
- Decisions affect the reward, the agent state, and environment state

This Lecture

- Last lecture: multiple actions, but only one state—no model
- This lecture:
  - Formalize 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

Formalizing the RL interface



- We discuss a mathematical formulation of the agent-environment interaction
- This is called a Markov decision process
- We can then talk clearly about the objective and how to reach it

Introduction to MDPs

- Markov decision processes (MDPs) formally describe an environment
- For now, assume the environment is fully observable:
  the current observation contains all relevant information
- Almost all RL problems can be formalized 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 Property

"The future is independent of the past given the present"
Consider a sequence of random states, $\{S_t\}_{t \in \mathbb{N}}$, indexed by time.

Definition

A state s has the Markov property when for states $\forall s' \in \mathcal{S}$ and all rewards $r \in \mathbb{R}$

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

for all possible histories $S_1, \ldots, S_{t-1}$

- 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

- 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
- p defines the dynamics of the problem
- The rewards and discount together define the goal
- Sometimes it is useful to marginalize out the state transitions or expected reward:

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

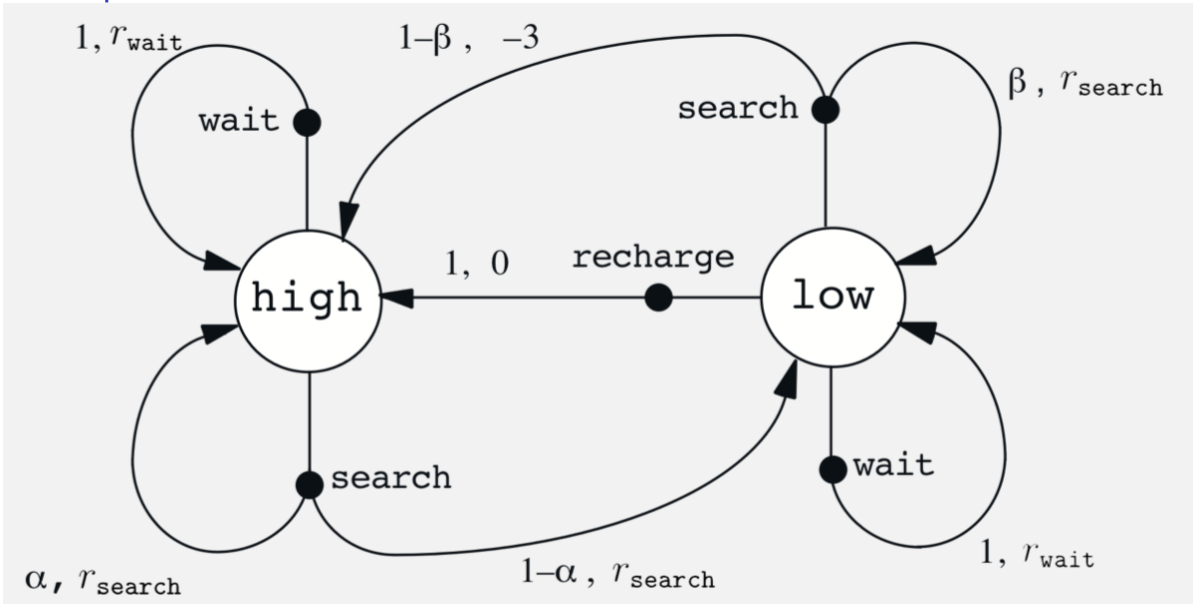
Example: cleaning robot

- Consider a robot that cleans 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)

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



Return

- Acting in a MDP results in returns G_t: total discounted reward from time-step t

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

- This is a random variables that depends on MDP and policy
- 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 reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Immediate rewards may actually be more valuable (e.g., consider earning interest)
- Animal/human behaviour shows preference for immediate reward
- Sometimes we can use undiscounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate
- Note that reward and discount together determine the goal

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 Equation in Matrix Form

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

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

where

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

Bellman Equation in Matrix Form

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

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

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

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

- 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

Optimal Value Function

Definition

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

Value Function

- Estimating v_π or q_π is called policy evaluation or, simply, prediction
- Estimating v_* or q_* is sometimes called control, because these can be used for policy optimization

Bellman equations

- There are four main Bellman equations:

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

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

Bellman equations

- There are equivalences between state and action values

$$v_\pi(s) = \sum_a \pi(a \mid s) q_\pi(s, a)$$

$$v_*(s) = \max_a q_*(s, a)$$

Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \quad \Longleftrightarrow \quad v^{\pi}(s) \geq v^{\pi'}(s) \ , \ \forall s$$

Theorem

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 $q^*(s, a)$,

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

- 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)

Solving the Bellman Optimality Equation

- The Bellman optimality equation is non-linear
- Cannot use the same direct matrix solution as for policy evaluation (in general)
- Many iterative solution methods
- Using models / dynamic programming
  - Value iteration
  - Policy iteration
- Using samples
  - Monte Carlo
  - Q-learning
  - Sarsa

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, emphasis mine)

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

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

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

Policy evaluation

- Does policy evaluation always converge?
- Answer: yes, under appropriate conditions (e.g., $\gamma < 1$)
- Simple proof-sketch (continuing, discounted case):

$$\begin{aligned}
\max_s |v_{k+1}(s) - v_\pi(s)| &= \max_s |\mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, \pi\right] \\
&\quad - \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, \pi\right]| \\
&= \max_s |\mathbb{E}\left[\gamma v_k(S_{t+1}) - \gamma v_\pi(S_{t+1}) \mid S_t = s, \pi\right]| \\
&= \gamma \max_s |\mathbb{E}\left[v_k(S_{t+1}) - v_\pi(S_{t+1}) \mid S_t = s, \pi\right]| \\
&\leq \gamma \max_s |v_k(s) - v_\pi(s)|
\end{aligned}$$

- Implies $\lim_{k\to\infty} v_k = v_\pi$
- Finite-horizon episodic case is a bit harder, but also works

Policy evaluation

actions

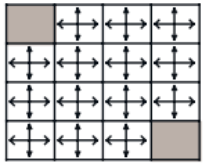
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$$R_t = -1$$
on all transitions

Policy evaluation

$k = 0$

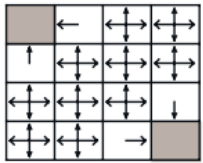
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

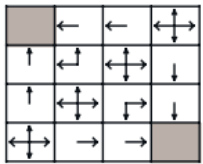
$k = 1$

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



$k = 2$

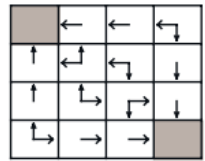
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Policy evaluation

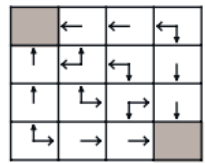
$k = 3$

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



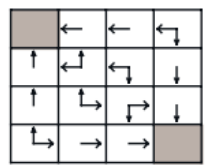
$k = 10$

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



optimal policy

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
- Instead, we can iterate, using

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

- Then, evaluate π_{new} and repeat
- One can show that $v_{\pi_{\text{new}}}(s) \geq v_\pi(s)$, for all s

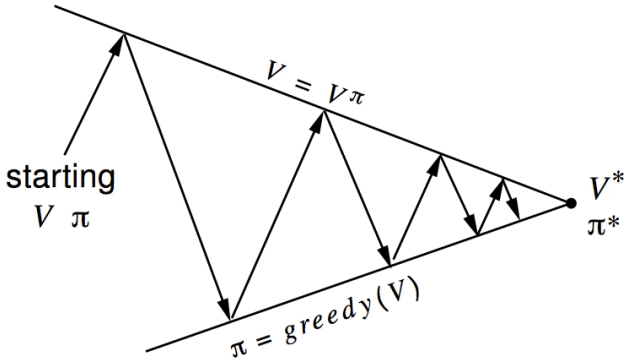
Policy Improvement

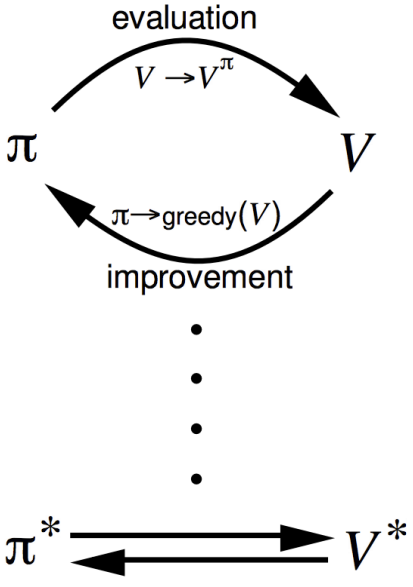
- One can show that $v_{\pi_{\text{new}}}(s) \geq v_{\pi}(s)$, for all s
- Let's assume that, at some point, we have $v_{\pi_{\text{new}}}(s) = v_{\pi}(s)$
- That implies

$$v_{\pi_{\text{new}}}(s) = \max_{a} \mathbb{E}\left[R_{t+1} + \gamma v_{\pi_{\text{new}}}(S_{t+1}) \mid S_t = s\right]$$

- But that is the Bellman optimality equation!
- Therefore, π_{new} must either be an improvement (when $\pi_{\text{new}} > \pi$), or be optimal (when $\pi_{\text{new}} = \pi$)

Policy Iteration





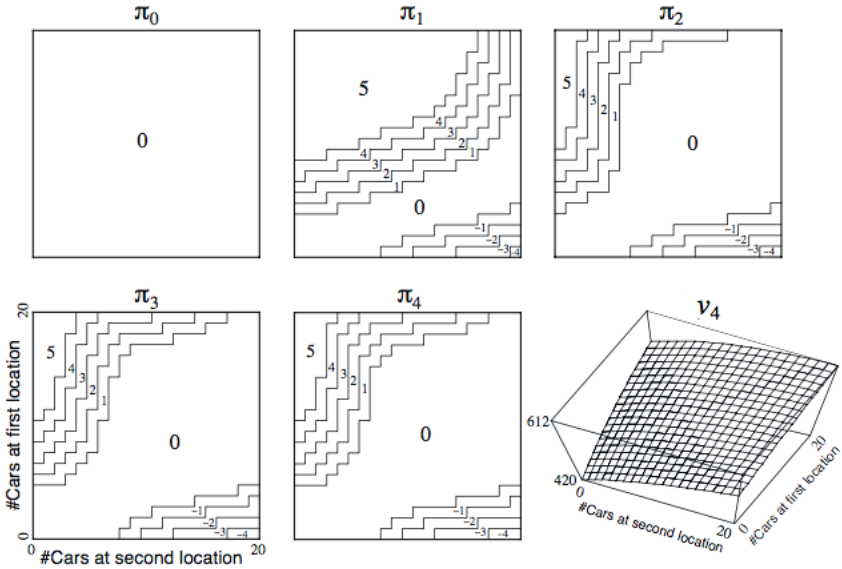
Policy evaluation Estimate v^{π}

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

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 $= 2$

Jack's Car Rental



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
- Why not update policy every iteration — i.e. stop after $k = 1$?
  - This is equivalent to value iteration (up next)

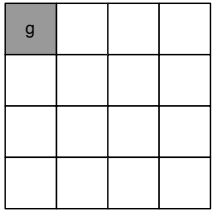
Value Iteration

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

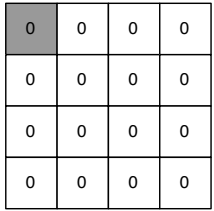
$$\forall s: \quad v_{k+1}(s) \leftarrow \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = s\right]$$

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

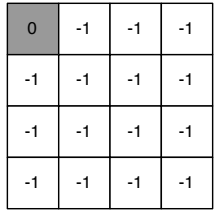
Example: Shortest Path



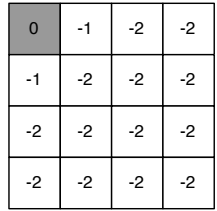
Problem



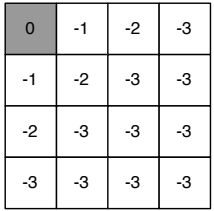
V_1



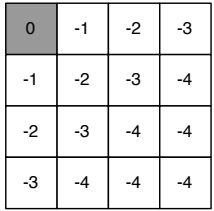
V_2



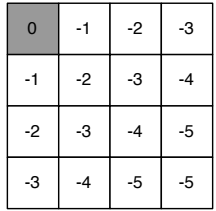
V_3



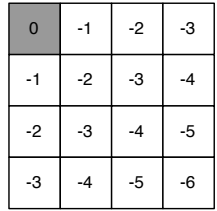
V_4



V_5



V_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

- Algorithms are based on state-value function $v^{\pi}(s)$ or $v^{*}(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q^{\pi}(s, a)$ or $q^{*}(s, a)$
- Complexity $O(m^2 n^2)$ per iteration

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.

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

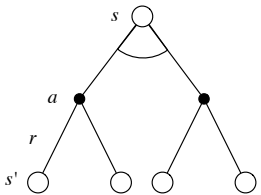
- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

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



Approximate Dynamic Programming

- Key idea: Approximate the value function
- Using a function approximator $v_\theta(s)$, with a parameter vector $\theta \in \mathbb{R}^m$
- The estimated value function at iteration k is v_{θ_k}
- Use dynamic programming to compute $v_{\theta_{k+1}}$ from v_{θ_k}
- e.g. 'fitted value iteration' repeats at each iteration k,
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$ (more simply, let $\tilde{\mathcal{S}} = \mathcal{S}$)
  - For each sampled state $s \in \tilde{\mathcal{S}}$, compute target using Bellman optimality equation,

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

  - Find θ_{k+1} by minimizing loss (e.g., with one gradient step)

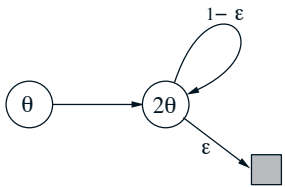
$$\sum_{s \in \tilde{\mathcal{S}}} \left(v_{\theta_k}(s) - \tilde{v}_k(s)\right)^2$$

Bootstrapping in Dynamic Programming

- Dynamic programming improves the estimate of the value at a state using the estimate of the value function at subsequent states
- This idea is core to RL — it is called bootstrapping
- There is a *theoretical* danger of divergence when combining
  1. Bootstrapping
  2. General function approximation
  3. Updating values for a state distribution that doesn't match the transition dynamics of the MDP
- This theoretical danger is rarely encountered in practice

Example of divergence with dynamic programming

- Tsitsiklis and Van Roy made an example where dynamic programming with linear function approximation can diverge. Consider the two state example below, where the rewards are all zero, there are no decisions, and there is a single parameter for estimating the value.



$$\begin{aligned}\theta_{k+1} &= \operatorname*{argmin}_{\theta} \sum_{s\in\mathcal{S}} (v_\theta(s) - \mathbb{E}\left[v_{\theta_k}(S_{t+1}) \mid S_t = s\right])^2 \\ &= \operatorname*{argmin}_{\theta} (\theta - \gamma 2\theta_k)^2 + (2\theta - \gamma(1-\epsilon)2\theta_k)^2 \\ &= \frac{6-4\epsilon}{5}\gamma\theta_k\end{aligned}$$

- What is $\lim_{k\to\infty} \theta_k$ when $\theta_0 = 1$, $\epsilon = \frac{1}{8}$, and $\gamma = 1$?
- This is only a problem when we update the states, e.g., synchronously, without looking at the time an agent would spend in each state

Questions?

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

For questions that arise outside of class, please use Moodle