CSC415: Introduction to Reinforcement Learning

Lecture 2: Planning by Dynamic Programming

Dr. Amey Pore

Winter 2026

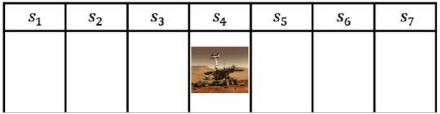
January 14, 2026

Structure and content adapted from David Silver's and Emma Brunskill's course on Introduction to RL.

Think Pair Wise 1

In a Markov decision process, a large discount factor γ means that short term rewards are much more influential than long term rewards.

Answers: • True • False • Don't know



- 7 discrete states (location of rover)
- 2 actions: Left or Right
- How many deterministic policies are there?

Outline

1. Recap
2. MDPs
3. Introduction to Dynamic Programming
4. Policy Iteration
5. Course logistics
6. Value Iteration

Return & Value Function

- **Definition of Horizon (H)**
  - Number of time steps in each episode
  - Can be infinite
  - Otherwise called **finite** Markov reward process
- **Definition of Return, G_t (for a MRP)**
  - Discounted sum of rewards from time step t to horizon H

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1}$$

- **Definition of State Value Function, $V(s)$ (for a MRP)**
  - Expected return from starting in state s

$$V(s) = \mathbb{E}[G_t | S_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1} | S_t = s]$$

Bellman Equation for MRPs

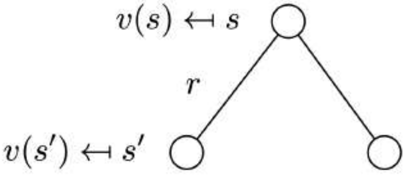
The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma V(S_{t+1})$

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \ldots) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]
\end{aligned}
$$

Bellman Equation for MRPs (2)

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$$



$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future rewards}}$$

Matrix Form of Bellman Equation for MRP

For finite state MRP, we can express $V(s)$ using a matrix equation

$$\begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix} = \begin{pmatrix} R(s_1) \\ \vdots \\ R(s_N) \end{pmatrix} + \gamma \begin{pmatrix} P(s_1|s_1) & \cdots & P(s_N|s_1) \\ P(s_1|s_2) & \cdots & P(s_N|s_2) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N) & \cdots & P(s_N|s_N) \end{pmatrix} \begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix}$$

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P} \mathbf{V}$$

Analytic Solution for Value of MRP

For finite state MRP, we can express $V(s)$ using a matrix equation

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P}\mathbf{V}$$

$$\mathbf{V} - \gamma \mathbf{P}\mathbf{V} = \mathbf{R}$$

$$(\mathbf{I} - \gamma \mathbf{P})\mathbf{V} = \mathbf{R}$$

$$\mathbf{V} = (\mathbf{I} - \gamma \mathbf{P})^{-1}\mathbf{R}$$

- Solving directly requires taking a matrix inverse $\sim O(N^3)$
- Requires that $(\mathbf{I} - \gamma \mathbf{P})$ is invertible
- Direct solutions only possible for small MRPs

Markov Decision Process (MDP)

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

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

- $\mathcal{S}$ is a (finite) set of Markov states $s \in \mathcal{S}$
- $\mathcal{A}$ is a (finite) set of actions $a \in \mathcal{A}$
- $\mathcal{P}$ is dynamics/transition model for **each action**, $P(s_{t+1} = s' | s_t = s, a_t = a)$
- $\mathcal{R}$ is a reward function $R(s_t = s, a_t = a) = \mathbb{E}[r_t | s_t = s, a_t = a]$
- Discount factor $\gamma \in [0, 1]$

Example: Mars Rover MDP

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

- 2 deterministic actions

$$P(s'|s,a_1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad P(s'|s,a_2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

MDP Policies

Definition

A *policy* π is a distribution over actions given states,

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

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent),

$$A_t \sim \pi(\cdot|S_t), \forall t > 0$$

MDP Policies (2)

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- The state sequence $S_1, S_2, \ldots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^{\pi} \rangle$
- The state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$
- where

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

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

Value Function under a Policy

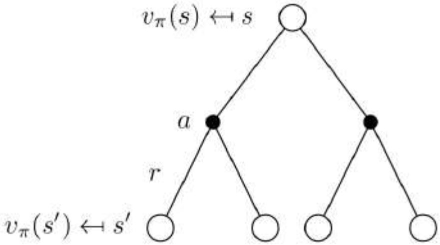
Definition

The state-value function $V^\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$$

Bellman Expectation Equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state, $V^{\pi}(s) = \mathbb{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s]$



$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi}(s') \right]$$

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 value function gives 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

$$\pi \geq \pi' \text{ if } V^{\pi}(s) \geq V^{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- There exists an optimal policy π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall \pi$
- All optimal policies achieve the optimal value function,

$$V^{\pi^*}(s) = V^*(s)$$

Finding an Optimal Policy

- Compute the optimal policy

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$$

- There exists a **unique** optimal value function
- Optimal policy for a MDP in an infinite horizon problem is deterministic

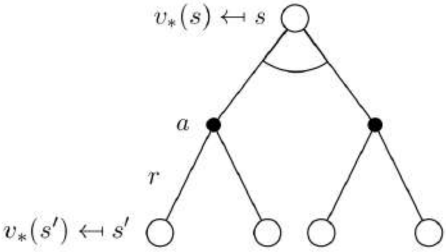
Finding an Optimal Policy

- Compute the optimal policy

$$\pi^*(s) = \arg\max_{\pi} V^{\pi}(s)$$

- There exists a **unique** optimal value function
- Optimal policy for a MDP in an infinite horizon problem (agent acts forever) is:
  - Deterministic
  - Stationary (does not depend on time step)
  - Unique? Not necessarily, may have two policies with identical (optimal) values

Bellman Optimality Equation



$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s')$$

Today's lecture: Recommendations

- The content is theoretical, but fundamental to RL
- Hence, if you don't ask questions, you will not understand
- You might get a reward for asking questions

What is Dynamic Programming?

- **Dynamic**: sequential or temporal component to the problem
- **Programming**: optimising a “program”, i.e. a policy
  - c.f. linear programming
- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

Requirements for Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

- **Optimal substructure**
  - Principle of optimality applies
  - Optimal solution can be decomposed into subproblems
- **Overlapping subproblems**
  - Subproblems recur many times
  - Solutions can be cached and reused
- **MDPs satisfy both properties**
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

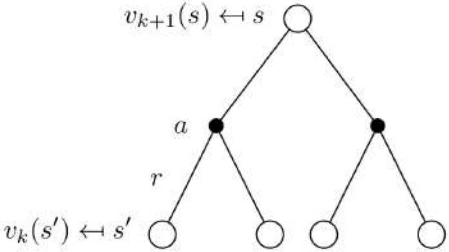
Planning by Dynamic Programming

- Dynamic programming assumes **full knowledge of the MDP**
- It is used for **planning** in an MDP
- **For prediction:**
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - or: MRP $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$
  - Output: value function V^{π}
- **Or for control:**
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function V^{*}
  - and: optimal policy π^{*}

Iterative Policy Evaluation

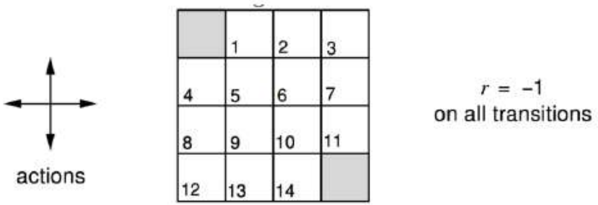
- **Problem**: evaluate a given policy π
- **Solution**: iterative application of Bellman expectation backup
- $V_1 \to V_2 \to \cdots \to V_\pi$
- Using **synchronous** backups,
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $V_{k+1}(s)$ from $V_k(s')$
  - where s' is a successor state of s

Iterative Policy Evaluation 2



$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_k(s') \right)$$

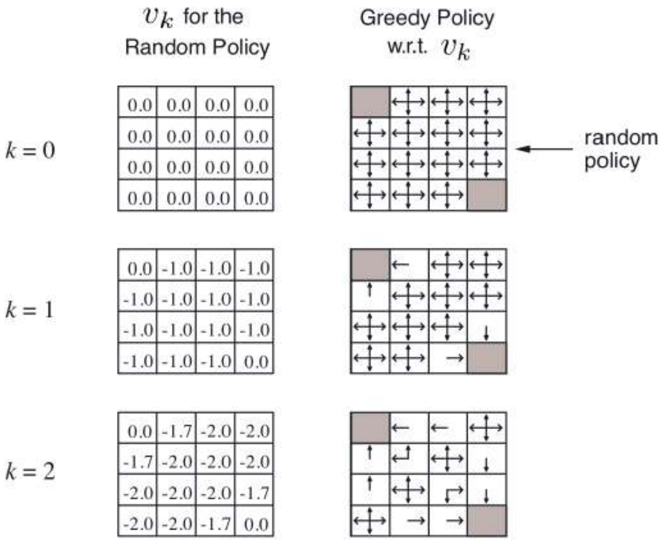
Evaluating a Random Policy in the Small Gridworld



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \ldots, 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

Iterative Policy Evaluation in Small Gridworld



Iterative Policy Evaluation in Small Gridworld (2)

$k = 3$

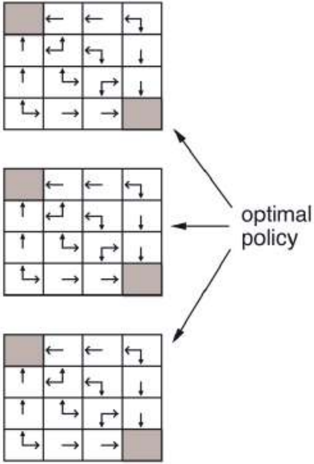
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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



Homework Exercise: MDP 1 Iteration of Policy Evaluation

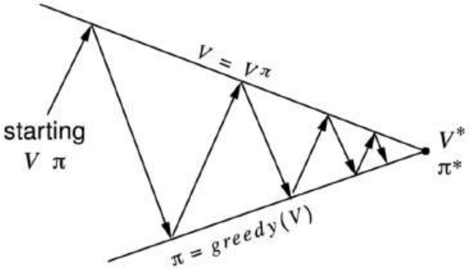
- **Dynamics:** $P(s_6|s_6, a_1) = 0.5, P(s_7|s_6, a_1) = 0.5, \dots$
- **Reward:** for all actions, $+1$ in state s_1, $+10$ in state s_7, 0 otherwise
- Let $\pi(s) = a_1 \forall s$, assume $V_k = [1, 0, 0, 0, 0, 0, 10]$ and $k = 1, \gamma = 0.5$
- **Compute** $V_{k+1}(s_6)$

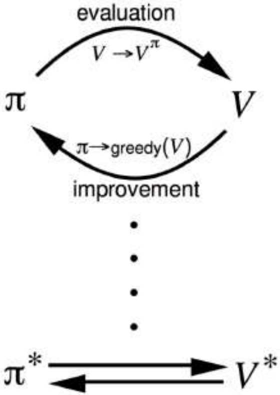
See answer at the end of the slide deck. Work this out yourself, if you'd like to practice. Then check the answer.

How to Improve a Policy

- Given a policy π
  - Evaluate the policy π
    $$V^{\pi}(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s]$$
  - Improve the policy by acting greedily with respect to V^{π}
    $$\pi' = \text{greedy}(V^{\pi})$$
- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to π^*

Policy Iteration





- Policy evaluation Estimate V^{π}
  Iterative policy evaluation
- Policy improvement Generate $\pi' \geq \pi$
  Greedy policy improvement

New Definition: State-Action Value Q

- State-action value of a policy

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi}(s')$$

- Take action a, then follow the policy π

Policy Improvement

- Compute state-action value of a policy π_i
  - For s in S and a in A:

$$Q^{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi_i}(s')$$

- Compute new policy π_{i+1}, for all $s \in S$

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a) \quad \forall s \in S$$

MDP Policy Iteration (PI)

- Set $i = 0$
- Initialize $\pi_0(s)$ randomly for all states s
- While $i == 0$ or $||\pi_i - \pi_{i-1}||_1 > 0$ (L1-norm, measures if the policy changed for any state):
  - $V^{\pi_i} \leftarrow$ MDP V function policy evaluation of π_i
  - $\pi_{i+1} \leftarrow$ Policy improvement
  - $i = i + 1$

Delving Deeper Into Policy Improvement Step

$$Q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi_i}(s')$$

Delving Deeper Into Policy Improvement Step

$$Q^{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi_i}(s')$$

$$\max_a Q^{\pi_i}(s,a) \geq R(s,\pi_i(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\pi_i(s)) V^{\pi_i}(s') = V^{\pi_i}(s)$$

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

- Suppose we take $\pi_{i+1}(s)$ for one action, then follow π_i forever
  - Our expected sum of rewards is at least as good as if we had always followed π_i
- But new proposed policy is to always follow π_{i+1} ...

Monotonic Improvement in Policy

$$\begin{aligned} V^{\pi_i}(s) &\leq \max_a Q^{\pi_i}(s,a) \\ &= \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_i}(s') \end{aligned}$$

Proof: Monotonic Improvement in Policy

$$
\begin{aligned}
V^{\pi_i}(s) &\le \max_a Q^{\pi_i}(s,a) \\
&= \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_i}(s') \\
&= R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s)) V^{\pi_i}(s') \quad \text{// by the definition of } \pi_{i+1} \\
&\le R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s)) \left(\max_{a'} Q^{\pi_i}(s',a') \right) \\
&= R(s,\pi_{i+1}(s)) + \gamma \sum_{s' \in S} P(s'|s,\pi_{i+1}(s)) \\
&\quad \left(R(s',\pi_{i+1}(s')) + \gamma \sum_{s'' \in S} P(s''|s',\pi_{i+1}(s')) V^{\pi_i}(s'') \right) \\
&\vdots \\
&= V^{\pi_{i+1}}(s)
\end{aligned}
$$

Course logistics

1. First lab tomorrow: dynamic programming, due date on Jan 20th at 11:59pm.
   - The submission will be on MarkUs.
   - Submit .ipynb notebook.
2. The office hours time for the TA has changed to Tuesday 5:15pm to 6:15pm.
   - Link to the zoom will be added on the home page Quercus.
3. Mid-term exam will take place on Jan 29th covering the first 4 lectures.
   - **Important:** We will need to exceed the tutorial time by 30mins. It is a 90 mins exam.
   - I will add sample questions on Piazza later today.

Think Pair Wise 2

- **If policy doesn't change, can it ever change again?**
  - Yes / No / Not sure
- **Is there a maximum number of iterations of policy iteration?**
  - Yes / No / Not sure

Break!

Outline

Principle of Optimality

Any optimal policy can be subdivided into two components:

- An optimal first action A_*
- Followed by an optimal policy from successor state S'

Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $V^\pi(s) = V^*(s)$, if and only if

- For any state s' reachable from s
- π achieves the optimal value from state s', $V^\pi(s') = V^*(s')$

Deterministic Value Iteration

- If we know the solution to subproblems $V^*(s')$
- Then solution $V^*(s)$ can be found by one-step lookahead

$$V^*(s) \leftarrow \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right]$$

- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards
- Still works with loopy, stochastic MDPs

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

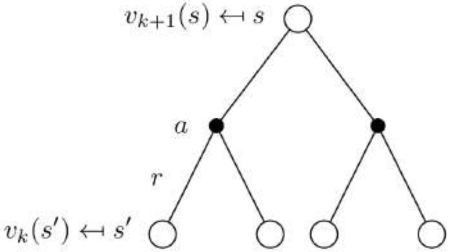
0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V^*$
- Using synchronous backups
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $V_{k+1}(s)$ from $V_k(s')$
- Convergence to V^* will be proven later
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

Value Iteration (2)



$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_k(s') \right]$$

Example of Value Iteration in Practice

`https://artint.info/demos/mdp/vi.html`

Going Back to Value Iteration (VI)

- Set $k = 1$
- Initialize $V_0(s) = 0$ for all states s
- Loop until convergence: (for ex. $||V_{k+1} - V_k||_\infty \leq \epsilon$)
  - For each state s

$$V_{k+1}(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \right]$$

- To extract optimal policy if can act for $k+1$ more steps,

$$\pi(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{k+1}(s') \right]$$

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 $O(m^2n^2)$ per iteration

Value vs Policy Iteration

	Policy Iteration	**Value Iteration**
Algorithm	Iterates through evaluation and improvement	Iterates through Bellman optimality updates
Updates	Solves V^{π} for a fixed policy	Updates $V(s)$ using max over a
Convergence	Fewer iterations, but each is costly	More iterations, but each is simple
Output	Directly improves the policy	Extracts policy from optimal values

What You Should Know

- Define MP, MRP, MDP, Bellman equation, model, Q-value, policy
- Be able to implement
  - Value Iteration
  - Policy Iteration
- Give pros and cons of different policy evaluation approaches
- Be able to prove contraction properties
- Limitations of presented approaches and Markov assumptions

Some Technical Questions

- How do we know that value iteration converges to V^*?
- Or that iterative policy evaluation converges to V^π?
- And therefore that policy iteration converges to V^*?
- Is the solution unique?
- How fast do these algorithms converge?
- These questions are resolved by *contraction mapping theorem*

Value Function Space

- Consider the vector space $\mathcal{V}$ over value functions
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a value function $V(s)$
- What does a Bellman backup do to points in this space?
- We will show that it brings value functions *closer*
- And therefore the backups must converge on a unique solution

Value Function ∞-Norm

- We will measure distance between state-value functions U and V by the ∞-norm
- i.e. the largest difference between state values,

$$\|U - V\|_\infty = \max_{s \in \mathcal{S}} |U(s) - V(s)|$$

Bellman Expectation Backup is a Contraction

- Define the *Bellman expectation backup operator* T^{π},

$$T^{\pi}(V) = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} V$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\begin{aligned}
\| T^{\pi}(U) - T^{\pi}(V) \|_{\infty} &= \| (\mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} U) - (\mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} V) \|_{\infty} \\
&= \| \gamma \mathcal{P}^{\pi} (U - V) \|_{\infty} \\
&\leq \| \gamma \mathcal{P}^{\pi} \|_{\infty} \| U - V \|_{\infty} \\
&\leq \gamma \| U - V \|_{\infty}
\end{aligned}$$

Contraction Mapping Theorem

Theorem (Contraction Mapping Theorem)

For any metric space $\mathcal{V}$ that is complete (i.e. closed) under an operator $T(V)$, where T is a γ-contraction,

- T converges to a unique fixed point
- At a linear convergence rate of γ

Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator T^{π} has a unique fixed point
- V^{π} is a fixed point of T^{π} (by Bellman expectation equation)
- By contraction mapping theorem
- Iterative policy evaluation converges on V^{π}
- Policy iteration converges on V^{*}

Bellman Optimality Backup is a Contraction

- Define the *Bellman optimality backup operator* T^*,

$$T^*(V) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a V$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ (similar to previous proof)

$$\|T^*(U) - T^*(V)\|_\infty \leq \gamma \|U - V\|_\infty$$

Convergence of Value Iteration

- The Bellman optimality operator T^* has a unique fixed point
- V^* is a fixed point of T^* (by Bellman optimality equation)
- By contraction mapping theorem
- Value iteration converges on V^*

Thank you!

Asynchronous Dynamic Programming

- DP methods described so far used *synchronous* backups
- i.e. all states are backed up in parallel
- *Asynchronous DP* backs up states individually, in any order
- For each selected state, apply the appropriate backup
- 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

  for all s in $\mathcal{S}$

  $v_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{old}(s') \right)$

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

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

  for all s in $\mathcal{S}$

  $v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right)$

Prioritised Sweeping

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

$$\left| \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right) - v(s) \right|$$

- 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

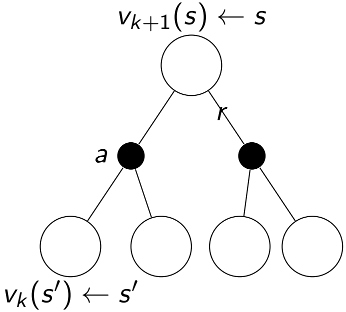
Real-Time Dynamic Programming

- Idea: only states that are relevant to agent
- Use agent's experience to guide the selection of states
- After each time-step S_t, A_t, R_{t+1}
- Backup the state S_t

$$v(S_t) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}^a_{S_t} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{S_t s'} v(s') \right)$$

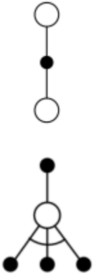
Full-Width Backups

- DP uses *full-width* backups
- For each backup (sync or async)
  - Every successor state and action is considered
  - Using knowledge of the MDP transitions and reward function
- DP is effective for medium-sized problems (millions of states)
- For large problems DP suffers Bellman's *curse of dimensionality*
  - Number of states $n = |\mathcal{S}|$ grows exponentially with number of state variables
- Even one 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 $\mathcal{R}$ and transition dynamics $\mathcal{P}$
- Advantages:
  - Model-free: no advance knowledge of MDP required
  - Breaks the curse of dimensionality through sampling
  - Cost of backup is constant, independent of $n = |\mathcal{S}|$



Approximate Dynamic Programming

- Approximate the value function
- Using a *function approximator* $\hat{v}(s, \mathbf{w})$
- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$
- e.g. Fitted Value Iteration repeats at each iteration k,
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
  - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,

    $\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \hat{v}(s', \mathbf{w}_k) \right)$
  - Train next value function $\hat{v}(\cdot, \mathbf{w}_{k+1})$ using targets $\{(s, \tilde{v}_k(s))\}$

Homework Solution

- **Dynamics:** $P(s_6|s_6, a_1) = 0.5$, $P(s_7|s_6, a_1) = 0.5$, ...
- **Reward:** for all actions, $+1$ in state s_1, $+10$ in state s_7, 0 otherwise
- Let $\pi(s) = a_1 \ \forall s$, assume $V_k = [1, 0, 0, 0, 0, 0, 10]$ and $k = 1$, $\gamma = 0.5$
- **Compute** $V_{k+1}(s_6)$

$$
\begin{aligned}
V_{k+1}(s_6) &= R(s_6) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s_6, a_1) V_k(s') \\
&= 0 + 0.5 \times (0.5 \times 10 + 0.5 \times 0) \\
&= 2.5
\end{aligned}
$$

Think pair wise 2: Explanation of Policy Not Changing

- Suppose for all $s \in \mathcal{S}$, $\pi_{i+1}(s) = \pi_i(s)$
- Then for all $s \in \mathcal{S}$, $Q^{\pi_{i+1}}(s, a) = Q^{\pi_i}(s, a)$

Recall policy improvement step:

- $Q^{\pi_i}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi_i}(s')$
- $\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s, a)$
- $\pi_{i+2}(s) = \arg\max_a Q^{\pi_{i+1}}(s, a) = \arg\max_a Q^{\pi_i}(s, a)$

Opportunities for Out-of-Class Practice

- Does the initialization of values in value iteration impact anything?
- Is the value of the policy extracted from value iteration at each round guaranteed to monotonically improve (if executed in the real infinite horizon problem), like policy iteration?