COMS W4701: Artificial Intelligence

Lecture 6: Dynamic Programming

Tony Dear, Ph.D.
Department of Computer Science
School of Engineering and Applied Sciences

Today

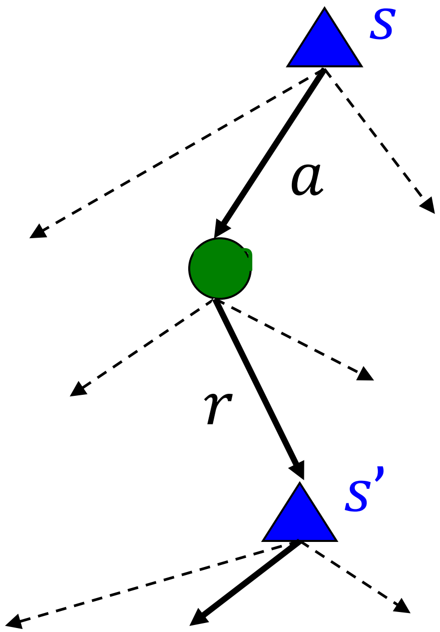
- Bellman optimality equations
- Dynamic programming for MDPs
- Value iteration
- Policy iteration

Markov Decision Processes

MDPs: Stochastic, sequential decision problems

- Set of states S, set of actions A
- Transitions $T(s, a, s') = \Pr(s'|s, a)$
- Rewards $R(s, a, s')$, discount γ

- We see state-action-reward $(s, a, r, s, a, r, \ldots)$ sequences



Can derive the following functions

- **Policy** $\pi: S \rightarrow A$, assignment of action to each state
- **Value** $V^{\pi}: S \rightarrow \mathbb{R}$, expected state utilities if following π

Recursive Relationship

$$V^{\pi}(s) = E\left[\sum_{t=0} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right]$$

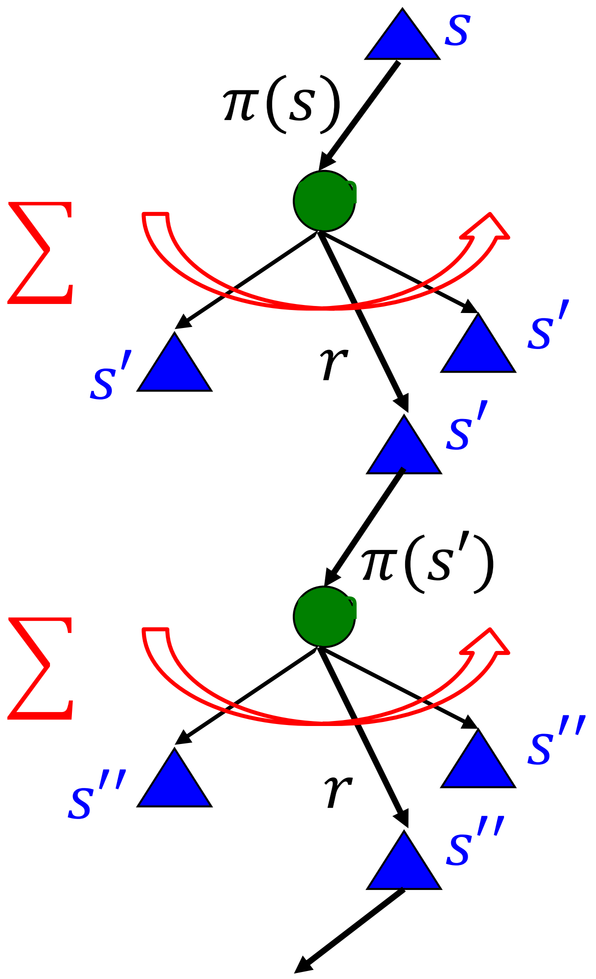
- This is a function on every state in the state space

- For a given state s, we can alternatively write $V^{\pi}(s)$ as a *recursive* function of *successor state values*

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

因为是recursively的 最终$V\pi$会被cancel

- $V^{\pi}(s)$ is a *weighted average* of (immediate reward plus discounted successor state values)



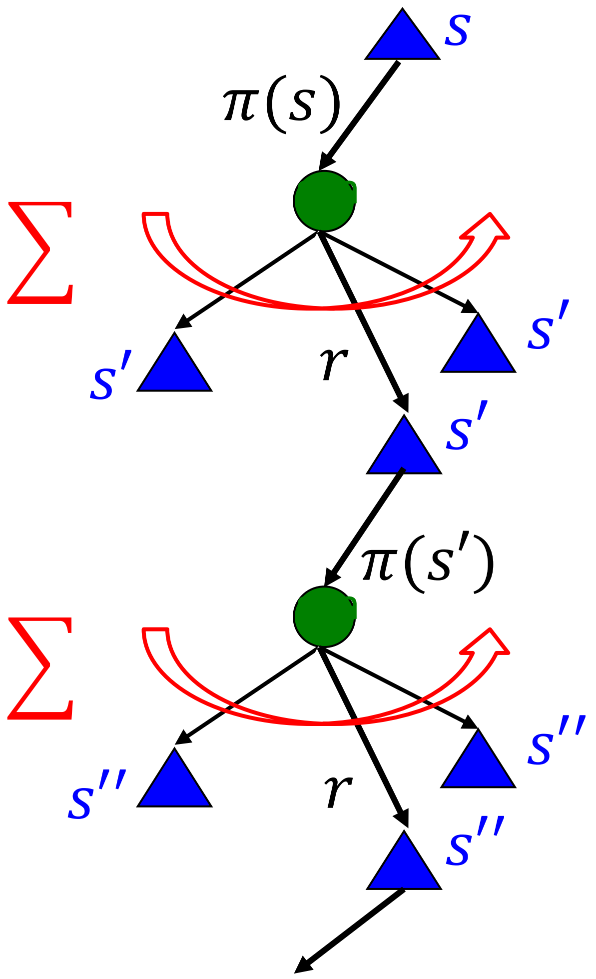
Solving for Values

$$V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

- Suppose we know the model (T, R), discount γ, and a fixed policy π

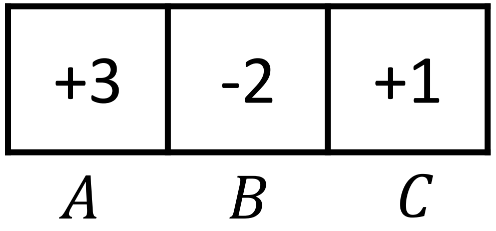
- Then the above is a system of $|S|$ *linear* equations in the $|S|$ unknowns $V^{\pi}(s)$

- Linear solvers: ~$O(|S|^3)$ time, can find $V^{\pi}(s)$



Example: Mini-Gridworld

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$



- States: A, B, C; actions: left, right; $\gamma = 0.5$
- Policy: $\pi(s) = \text{left} \ \forall \ s$
- Rewards: $R(s, a, A) = +3, R(s, a, B) = -2, R(s, a, C) = +1$
- Transitions: $\Pr(\text{intended direction}) = 0.8, \Pr(\text{opposite direction}) = 0.2$

- V^{π} can be found by solving 3 linear equations:

$$V^{\pi}(A) = 0.8\big(3 + 0.5V^{\pi}(A)\big) + 0.2(-2 + 0.5V^{\pi}(B))$$

$$V^{\pi}(B) = 0.8\big(3 + 0.5V^{\pi}(A)\big) + 0.2(1 + 0.5V^{\pi}(C))$$

$$V^{\pi}(C) = 0.8\big(-2 + 0.5V^{\pi}(B)\big) + 0.2(1 + 0.5V^{\pi}(C))$$

Bellman Optimality Equations

- Our goal is to find an **optimal policy** or **optimal value function**

$$V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

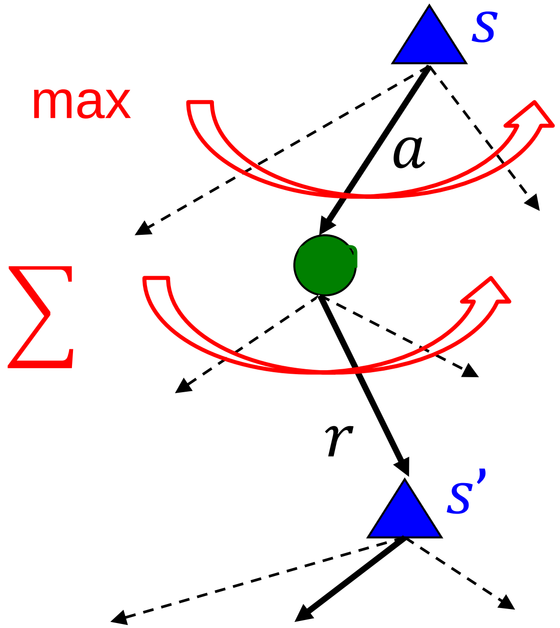
$$\pi^* = \operatorname{argmax}_{\pi} V^{\pi}$$

$$V^* = \max_{\pi} V^{\pi}$$

$$V^*(s) = \max_{a} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

$$\pi^*(s) = \operatorname*{argmax}_{a} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

对于每个state找best action



- **Bellman optimality equations** are nonlinear!

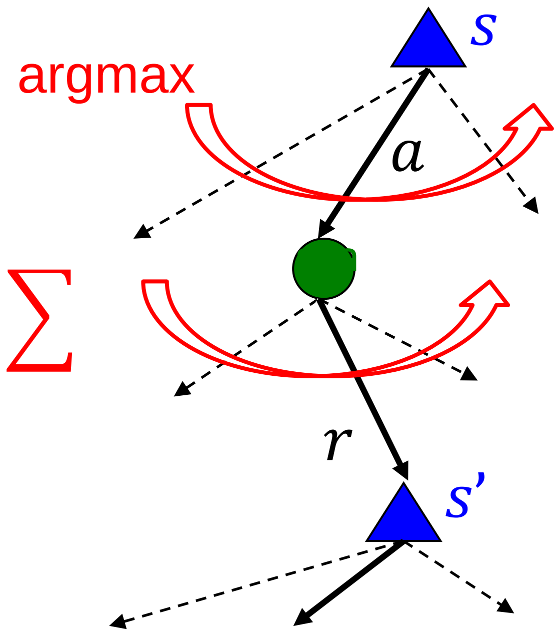
Value Function to Policy

- We don't know (yet) how to solve for V^* from scratch
- We *do* know how to find V^* given π^* (solve linear system)

- Bellman equation tells us how to find π^* given V^*

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- Everything on the RHS is known!
- Solving for complete policy takes $O(|S|^2|A|)$ time



Example: Mini-Gridworld

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

+3	-2	+1

- States: A, B, C; actions: left, right
- Transitions and rewards same as before

- Given $V^*(A) = 4.06, V^*(B) = 4.36, V^*(C) = 1.39$
- Find $\pi^*(B)$:

$\gamma = 0.5$

$$\pi^*(B) = \operatorname{argmax} \begin{cases} 0.8\big(3 + 0.5V^*(A)\big) + 0.2\big(1 + 0.5V^*(C)\big) & \text{Left} \\ 0.8\big(1 + 0.5V^*(C)\big) + 0.2(3 + 0.5V^*(A)) & \text{Right} \end{cases}$$

State-Action Values

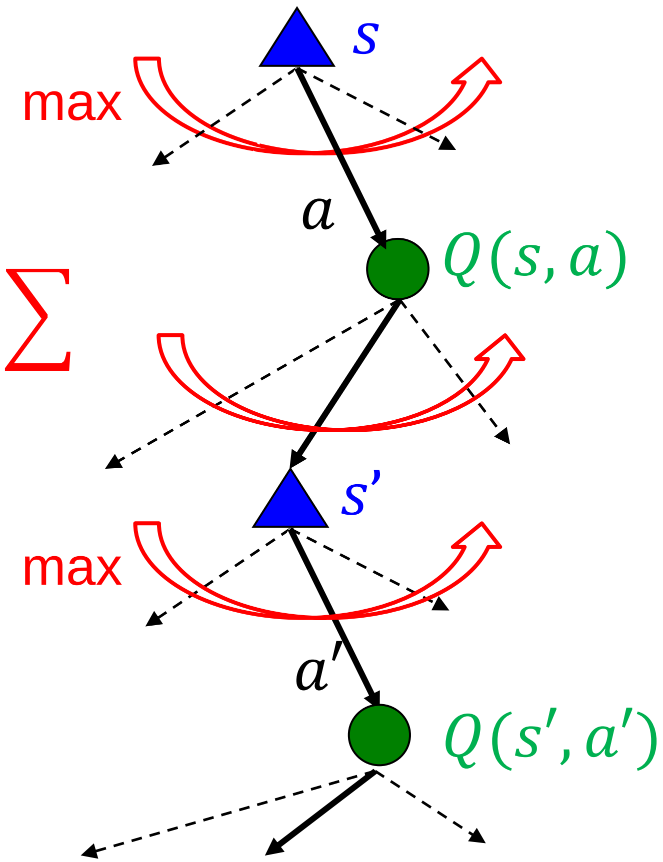
- We can also tidy up the Bellman equations by defining **state-action values** (Q-values) of **Q-states**

- Interpretation: Agent has committed to an action, but transition has not yet resolved

$$Q(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

$$= \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

$$V^*(s) = \max_a Q(s,a) \qquad \pi^*(s) = \operatorname{argmax}_a Q(s,a)$$



Solving the Bellman Equations

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- How to solve for V^*?
- We have $|S|$ nonlinear equations (because of max)!

- Dynamic programming: *Iterate* on **time-limited values** V_i

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$$

- Bellman equation rewritten as Bellman *update rule*

Value Iteration

Idea: Repeatedly applying *Bellman update* to approximations of value function brings it closer to optimal (true) values V^*

- Initialize: $V_0(s) \leftarrow 0$ for all states s
- **Loop** from $i = 0$:
  - **For** each state $s \in S$:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$$

- **Until** $\max_s |V_{i+1}(s) - V_i(s)| <$ *small threshold*

Example: Mini-Gridworld

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$$

+3	-2	+1

- States: A, B, C; actions: L, R
- Rewards received when entering state
- Transitions: $\Pr(\text{intended direction}) = 0.8, \Pr(\text{opposite direction}) = 0.2$

- Initialize: $\big(V_0(A), V_0(B), V_0(C)\big) = (0,0,0)$ $\qquad \gamma = 0.5$

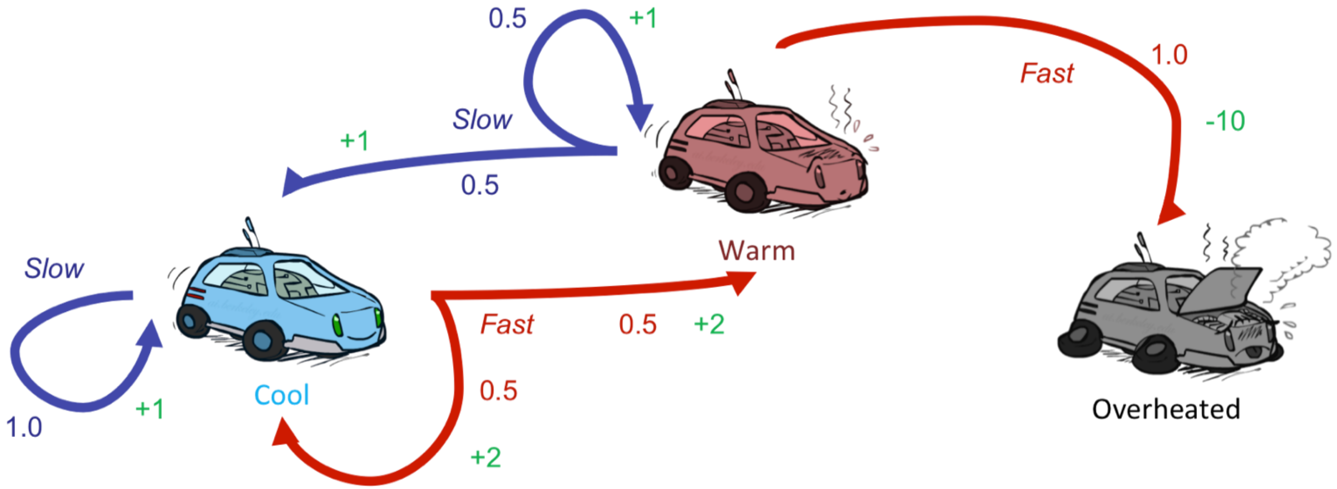
$$\begin{pmatrix} V_1(A) \\ V_1(B) \\ V_1(C) \end{pmatrix} = \begin{pmatrix} \max\big[0.8\big(3 + 0.5(0)\big) + 0.2\big(-2 + 0.5(0)\big), 0.8\big(-2 + 0.5(0)\big) + 0.2\big(3 + 0.5(0)\big)\big] \\ \max[\big(0.8\big(3 + 0.5(0)\big) + 0.2\big(1 + 0.5(0)\big), 0.8\big(1 + 0.5(0)\big) + 0.2\big(3 + 0.5(0)\big)\big] \\ \max[\big(0.8\big(-2 + 0.5(0)\big) + 0.2\big(1 + 0.5(0)\big), 0.8\big(1 + 0.5(0)\big) + 0.2\big(-2 + 0.5(0)\big)\big] \end{pmatrix} = \begin{pmatrix} 2 \\ 2.6 \\ 0.4 \end{pmatrix}$$

- $V_2, V_3, \ldots$ until convergence

Example: Race Car

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0

Assume no discount: $\gamma = 1$



$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

$V_2($$) = \max(1[1 + 1(2)], 0.5[2 + 1(2)] + 0.5[2 + 1(1)])$

$= \max(3, 3.5)$

$V_2($$) = \max(0.5[1 + 1(2)] + 0.5[1 + 1(1)], 1[-10 + 1(0)])$

$= \max(2.5, -10)$

Iterative Policy Evaluation

- Iterative update idea can also be used to solve for values of a fixed policy!
- Alternative to solving linear system; no max since actions are fixed

- Initialize $V_0(s)$ for all states s
- **Loop** from $i = 0$:
  - **For** each state $s \in S$:

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^{\pi}(s')]$$

- **Until** $\max_s |V_{i+1}(s) - V_i(s)| <$ *small threshold*

Convergence of Value Iteration

- The Bellman update is a **contraction mapping**

- Fact 1: Bellman update does not change optimal values V^* (*fixed point*)

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- Fact 2: The *max norm* $\|V_i - V^*\| = \max_s |V_i(s) - V^*(s)|$ between any value function V_i and V^* satisfies

$$\|V_{i+1} - V^*\| \leq \gamma \|V_i - V^*\|$$

- Each update shrinks "error" in V by factor of γ!

Rate of Convergence

- Value iteration converges *exponentially* fast

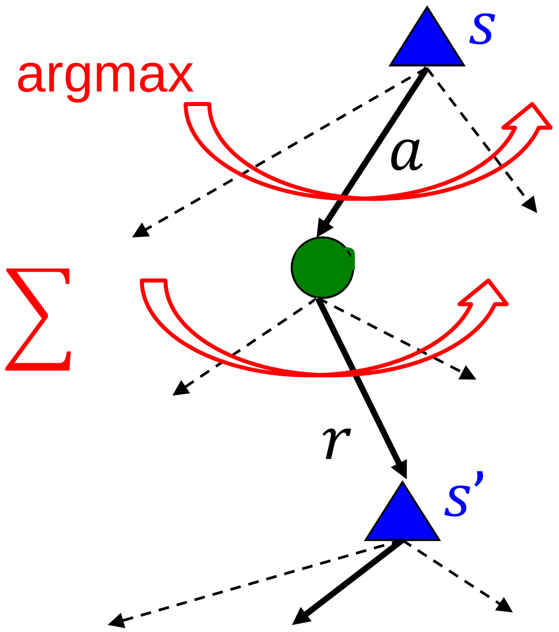
- Recall that a state's value is bounded by $\frac{|r_{\max}|}{1-\gamma}$
- After k passes of value iteration, error is bounded by $\gamma^k \frac{|r_{\max}|}{1-\gamma}$

- The smaller the γ, the faster that the error shrinks
- Tradeoff: Decisions become more myopic, future rewards less appealing

Values to Policy

- The goal of value iteration is to eventually extract an optimal policy
- We already know how to find π^* given V^*:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$



- Observation: We don't *require* optimal values
- Only relative values matter

- Policy may converge long before values do

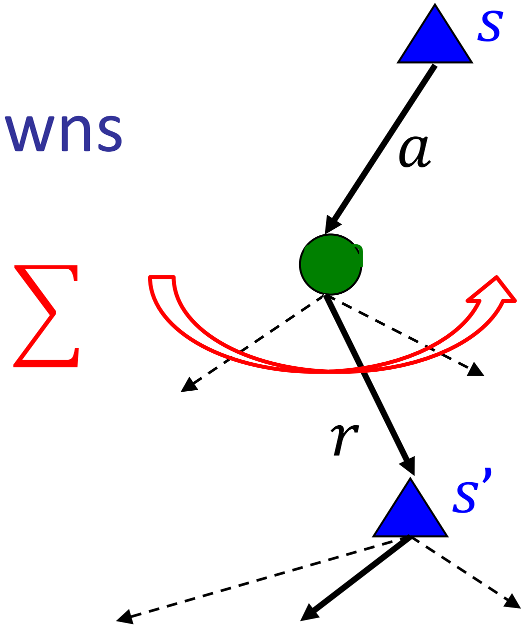
Policy Iteration

- Idea: A policy can be computed at any point during value iteration
- We can improve on policy *directly*, leading to better values, leading to a better policy, and so on…

- Initialize $\pi_1(s)$ arbitrarily, $V^{\pi_0}(s) \leftarrow 0$ for all states s
- **Loop** from $i = 1$:
  - **Policy evaluation**: Compute V^{π_i} *using* $V^{\pi_{i-1}}$ *as initial values*
  - **Policy improvement**: From V^{π_i}, find new policy π_{i+1}
- **Until** $\pi_{i+1} = \pi_i$

Policy Evaluation

$$V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$



- Review: System of $|S|$ linear equations in $|S|$ unknowns

- Alternatively, take an iterative approach:

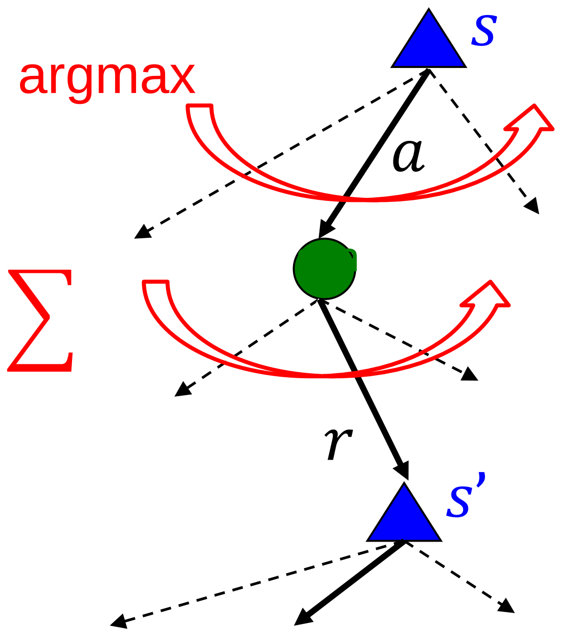
$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_{k}^{\pi}(s')]$$

- This is just value iteration without max!
- Can be faster if initialized with values of similar policy

Policy Improvement

- Given values for a fixed policy, how can we improve it?
- Consider taking "greediest" action at each state:

$$\pi_{i+1}(s) = \underset{a}{\text{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$$



- If π_i already optimal, then $V^{\pi_i} = V^*$ and $\pi_{i+1} = \pi_i$
- Otherwise, V^{π_i} can be moved closer to V^* by changing some actions
- Updating policy using argmax analogous to updating values using max

Example: Mini-Gridworld

+3	-2	+1

- States: A, B, C; actions: L, R
- Rewards received when entering state
- Transitions: $\Pr(\text{intended direction}) = 0.8, \Pr(\text{opposite direction}) = 0.2$

- Suppose we initialize $\left(\pi_0(A), \pi_0(B), \pi_0(C)\right) = (R, R, R)$
- **Evaluate** policy (either solve linear eqs or iterate Bellman-style):

$$\left(V_0(A), V_0(B), V_0(C)\right) = (-0.333, 1.75, 0.958)$$

$$\gamma = 0.5$$

- **Improve** policy:

$$\begin{pmatrix} \pi_1(A) \\ \pi_1(B) \\ \pi_1(C) \end{pmatrix} = \begin{pmatrix} \text{argmax}[0.8\left(3 + 0.5V_0(A)\right) + 0.2\left(-2 + 0.5V_0(B)\right), -0.333] \\ \text{argmax}[\left(0.8\left(3 + 0.5V_0(A)\right) + 0.2\left(1 + 0.5V_0(C)\right), 1.75\right] \\ \text{argmax}[\left(0.8\left(-2 + 0.5V_0(B)\right) + 0.2\left(1 + 0.5V_0(C)\right), 0.958\right] \end{pmatrix} = \begin{pmatrix} L \\ L \\ R \end{pmatrix}$$

Value Iteration vs Policy Iteration

$$
\begin{tikzcd}
V_0 \arrow[r] & V_1 \arrow[r] & V_2 \arrow[r] & \cdots \arrow[r] & V^* \arrow[d] \\
& & & & \pi^*
\end{tikzcd}
$$

$$
\begin{tikzcd}
V^{\pi_0} \arrow[dr] & V^{\pi_1} \arrow[dr] & V^{\pi_2} \arrow[dr] & \cdots \arrow[dr] & V^* \\
\pi_0 \arrow[u] & \pi_1 \arrow[u] & \pi_2 \arrow[u] & \cdots \arrow[u] & \pi^* \arrow[u, dotted]
\end{tikzcd}
$$

- Computes values only
- Keeps track of values only

- Each sweep consists of one iterative policy evaluation (sum) and policy improvement (max)

- Computes values and policy
- Keeps track of policy only

- Each sweep consists of many iterative policy evaluations (sum) and policy improvement (argmax)

Algorithm Complexity

- Each sweep of value iteration takes $O(|S|^2|A|)$ time

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$$

- Each sweep of policy iteration takes $O(|S|^3 + |S|^2|A|)$ time

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s')[R(s,\pi_i(s),s') + \gamma V_k^{\pi_i}(s')]$$

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$$

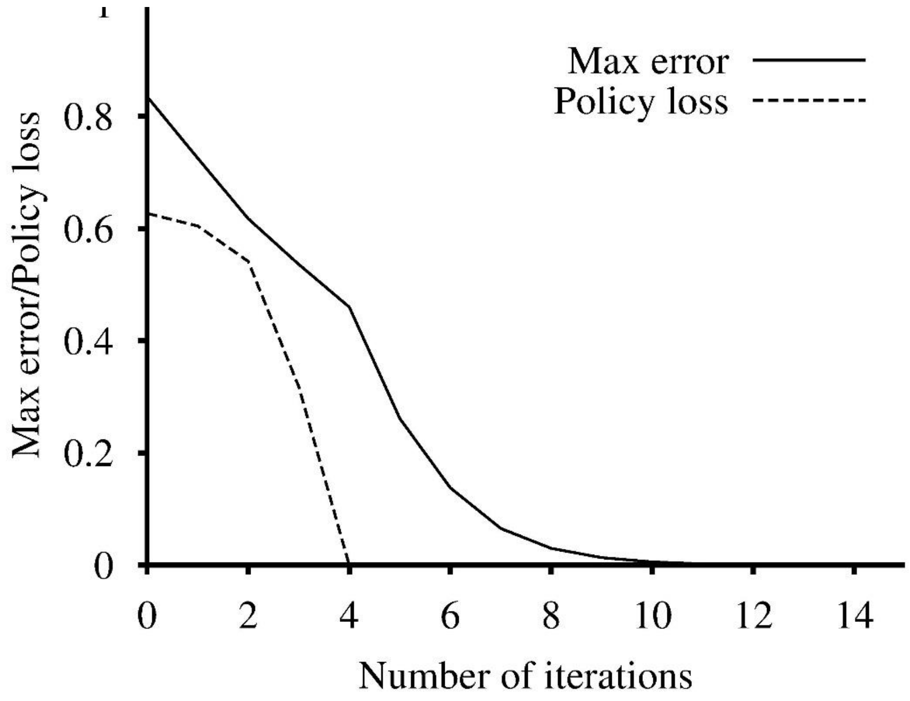
- In practice, complexity is also strongly dependent on problem at hand

Algorithm Complexity

- Value iteration: $O(|S|^2|A|)$
- Policy iteration: $O(|S|^3 + |S|^2|A|)$

- Value iteration: Number of sweeps depends on γ and error threshold
- Increases dramatically for high discount factor

- Policy iteration: Policy evaluation typically much more efficient than $O(|S|^3)$
- Fewer sweeps needed overall to converge



Summary

- Dynamic programming solves MDPs exactly by using recursive relationships among the state values
- Bellman updates push values and policies toward the optimal solution

- Value iteration: Compute and converge toward optimal values for all states, then extract policy

- Policy iteration: Alternate between evaluating a current policy and improving the policy