COMS W4701: Artificial Intelligence

Lecture 4b: Dynamic Programming

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

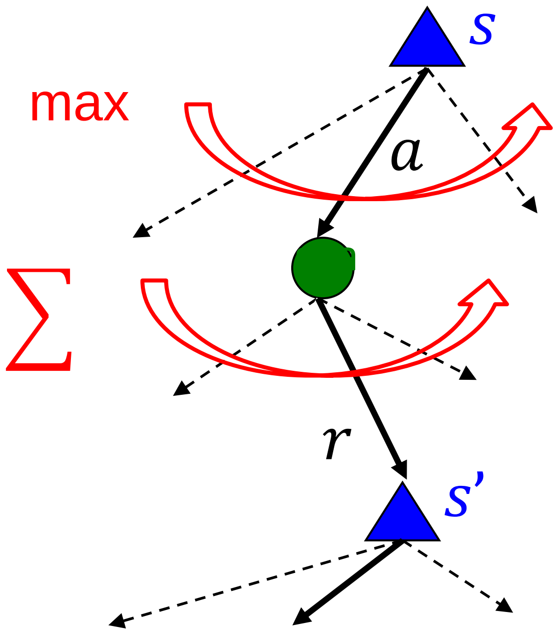
Today

- Bellman optimality equations
- Time-limited values
- Iterative policy evaluation
- Value iteration

Bellman Optimality Equations

- Generally, we want to find an **optimal policy** or **optimal value function**

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



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

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

Bellman optimality equations

Bellman Optimality Equations

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

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

- The Bellman optimality equations are *nonlinear*
- We cannot solve a system of linear equations to find an optimal policy

- Assuming we *can* solve for V^*, it is feasible to find π^* using a brute force search over all actions at each state and taking the argmax

Time-Limited Values

- Math problem: State values are nonlinear functions of other state values
- We can try traversing the tree using search, but may be exponentially large

- Idea from depth-limited search: Treat non-terminal states as terminal
- If the time horizon is 0, *every* state is effectively “terminal”

- **Time-limited values** $V_t(s)$: Expected utility of t decisions starting from s
- If we have $V_t(s)$, we can perform a tree value backup to compute $V_{t+1}(s)$

Time-Limited Values

- Given a policy π, we can *iterate* over time-limited values to solve for V^{π}
- Alternative method to fully solving a system of linear equations

- Base case: Time horizon $i = 0$. No more rewards, so $V_0(s) = 0\ \forall s$

- $i = 1$: $V_1^{\pi}(s) = \sum_{s'} T(s, \pi(s), s') R(s, \pi(s), s')$
- $i = 2$: $V_2^{\pi}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_1^{\pi}(s')]$
- $i = 3$: $V_3^{\pi}(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_2^{\pi}(s')]$
- …

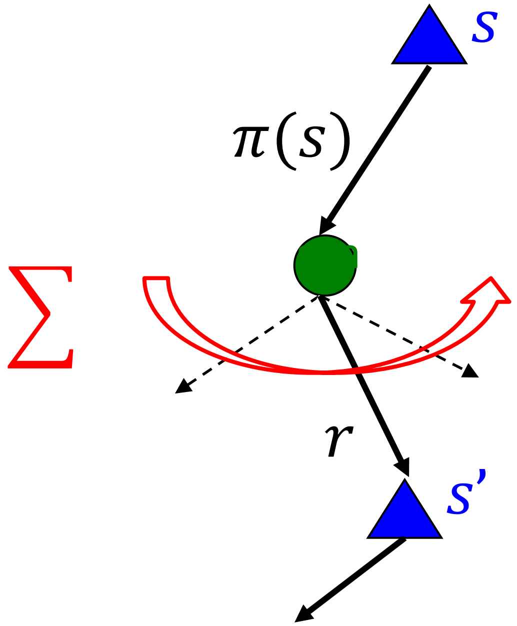
Iterative Policy Evaluation

- Using V_i to iteratively compute V_{i+1} is an example of *dynamic programming*

- Initialize $V_0^\pi(s) \leftarrow 0$ for all states s
- **Loop** from $i = 0$:
  - **Initialize** temporary array V_{i+1}
  - **For** each state $s \in S$:

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

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



Example: Mini-Gridworld

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

+3	-2	+1
A	B	C

- Suppose we are given the policy $\pi(s) = L\ \forall s$
- Suppose we use the discount factor $\gamma = 0.5$

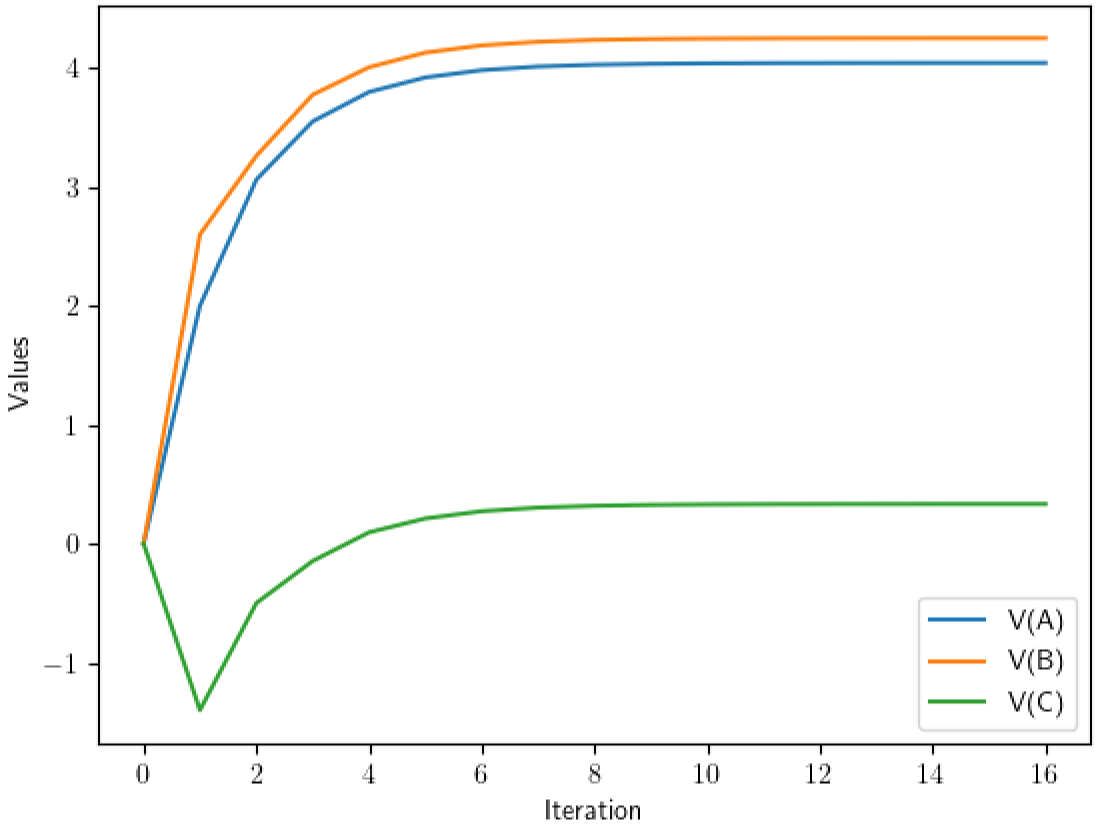
- We can iteratively perform three value updates, one for each state:

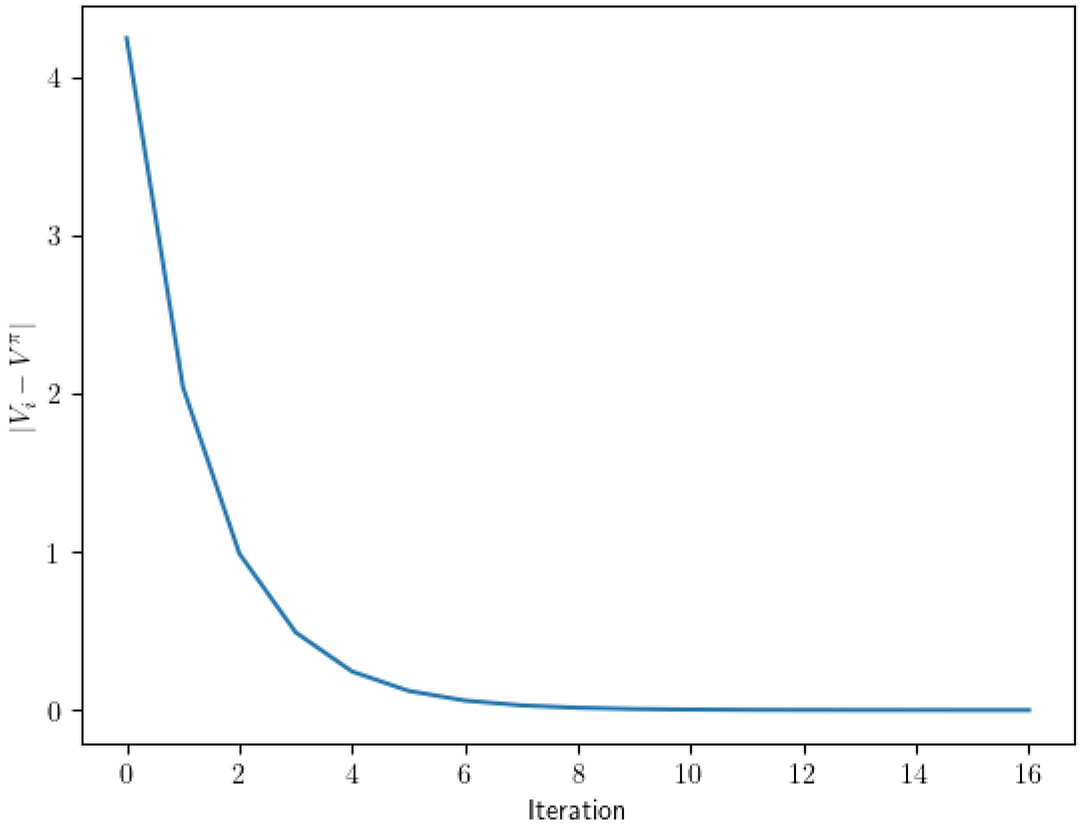
$$V_{i+1}^{\pi}(A) = 0.8\big(3 + 0.5V_i^{\pi}(A)\big) + 0.2(-2 + 0.5V_i^{\pi}(B))$$

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

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

Example: Mini-Gridworld





Value Iteration

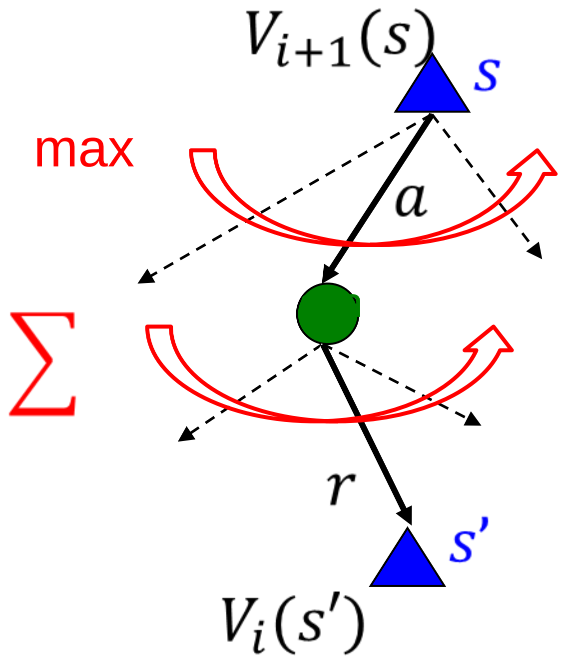
- Now suppose we want to find V^* rather than evaluate a fixed policy
- Additional step: Find the value of the *best* action in each iteration

- Initialize: $V_0(s) \leftarrow 0$ for all states s
- **Loop** from $i = 0$:
  - **Initialize** temporary array V_{i+1}
  - **For** each state $s \in S$:

Bellman update

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

  - **Copy** V_{i+1} into V_i
- **Until** $\max_{s} |V_{i+1}(s) - V_i(s)| < \epsilon$ (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
A	B	C

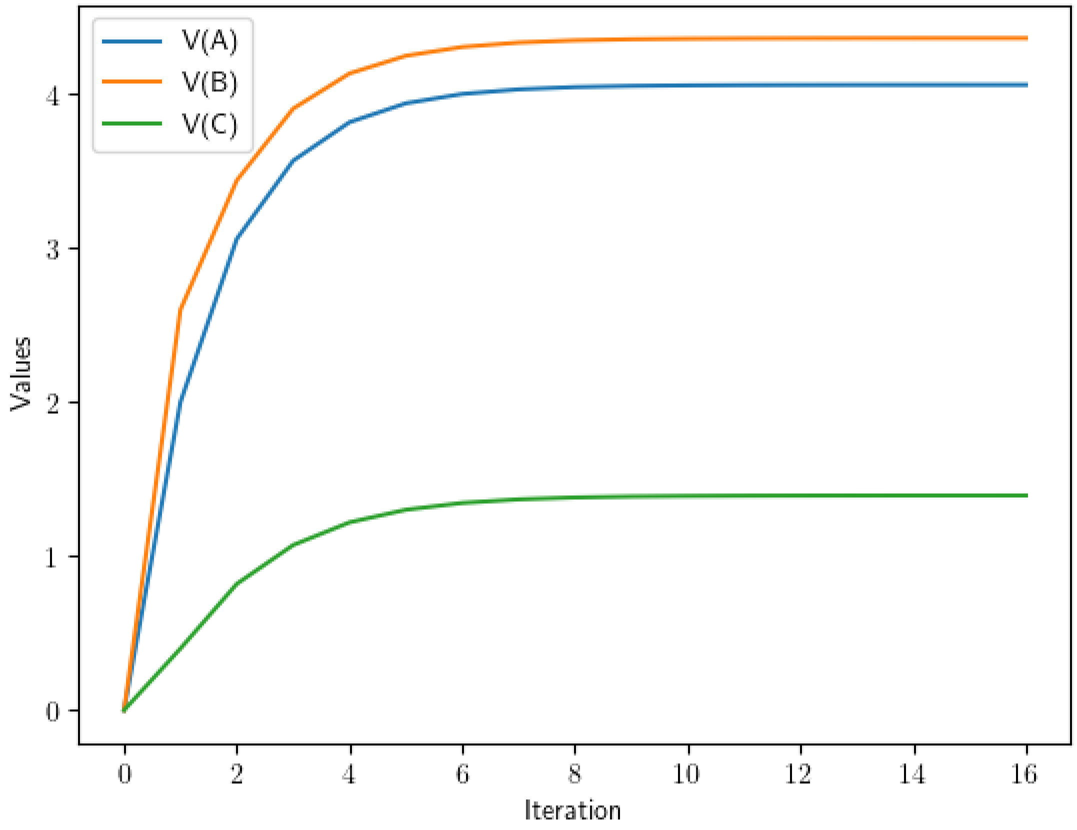
- States, actions, rewards as shown; $\gamma = 0.5$
- Transitions: $\mathrm{Pr}(\text{intended direction}) = 0.8$, $\mathrm{Pr}(\text{opposite direction}) = 0.2$

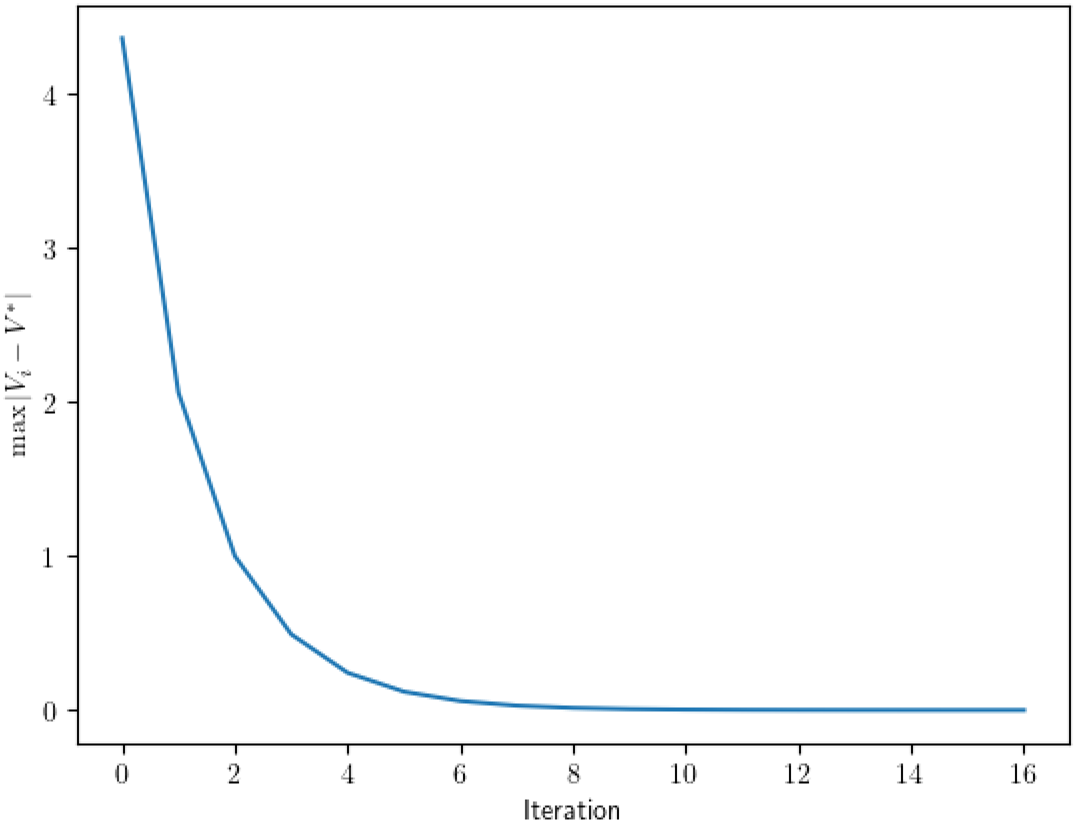
$$V_{i+1}(A) = \max \begin{Bmatrix} 0.8\big(3 + 0.5V_i(A)\big) + 0.2\big(-2 + 0.5V_i(B)\big), & L \text{ action} \\ 0.8\big(-2 + 0.5V_i(B)\big) + 0.2\big(3 + 0.5V_i(A)\big) & R \text{ action} \end{Bmatrix}$$

$$V_{i+1}(B) = \max \begin{Bmatrix} 0.8\big(3 + 0.5V_i(A)\big) + 0.2\big(1 + 0.5V_i(C)\big), & L \text{ action} \\ 0.8\big(1 + 0.5V_i(C)\big) + 0.2\big(3 + 0.5V_i(A)\big) & R \text{ action} \end{Bmatrix}$$

$$V_{i+1}(C) = \max \begin{Bmatrix} 0.8\big(-2 + 0.5V_i(B)\big) + 0.2\big(1 + 0.5V_i(C)\big), & L \text{ action} \\ 0.8\big(1 + 0.5V_i(C)\big) + 0.2\big(-2 + 0.5V_i(B)\big) & R \text{ action} \end{Bmatrix}$$

Example: Mini-Gridworld





Convergence of Value Iteration

- The Bellman update is a **contraction mapping**
- Time-limited value are always guaranteed to move toward V^*

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

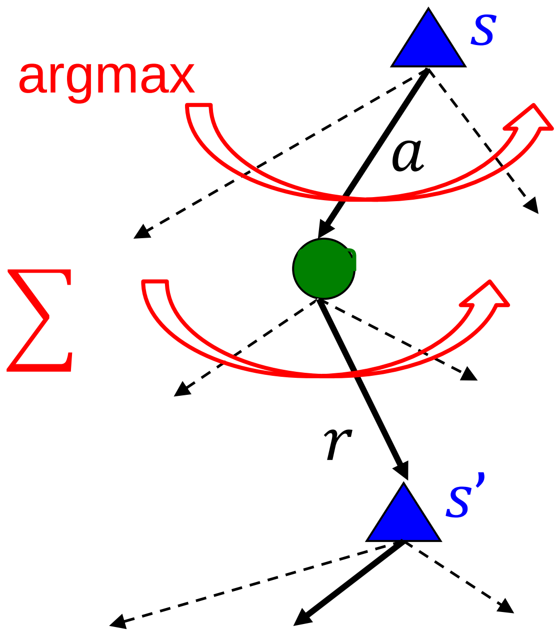
$$V^*(s) = \max_a \sum\nolimits_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- Define the *max norm* (error) $\|V_i - V^*\| = \max_s |V_i(s) - V^*(s)|$
- Fact 2: Each update shrinks max error in V by factor of γ: $\|V_{i+1} - V^*\| \leq \gamma \|V_i - V^*\|$
- Errors decrease (and values converge) exponentially fast!

Policy Extraction

- How do we back out π^* after computing V^*?
- (Recursive) definition of π^* from Bellman equation:

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



- Everything on the RHS is now known!

- For each state, iterate through all actions
- Optimal policy assigns action with the highest utility

Algorithm Complexity

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

- Each sweep of value iteration involves, at each state, an expectation over all successor states and a maximization over all actions
- Same operation for policy extraction, so each sweep takes $O(|S|^2|A|)$ time

- The *number* of sweeps depends on discount factor γ and error threshold ϵ
- Since min error reduction is γ per iteration, a forward-looking agent (larger γ) requires more sweeps before convergence

Summary

- Dynamic programming solves MDPs using time-limited quantities derived from decision making in finite time horizons
- Bellman updates push values and policies toward the optimal solution
- Policy evaluation: Iteratively update and solve for values of fixed policy
- Value iteration: Compute optimal values for all states by iteratively finding best actions and their values at each iteration