

Model Based Control: Policy Iteration and Value Iteration

Lecture 3
Subir Varma

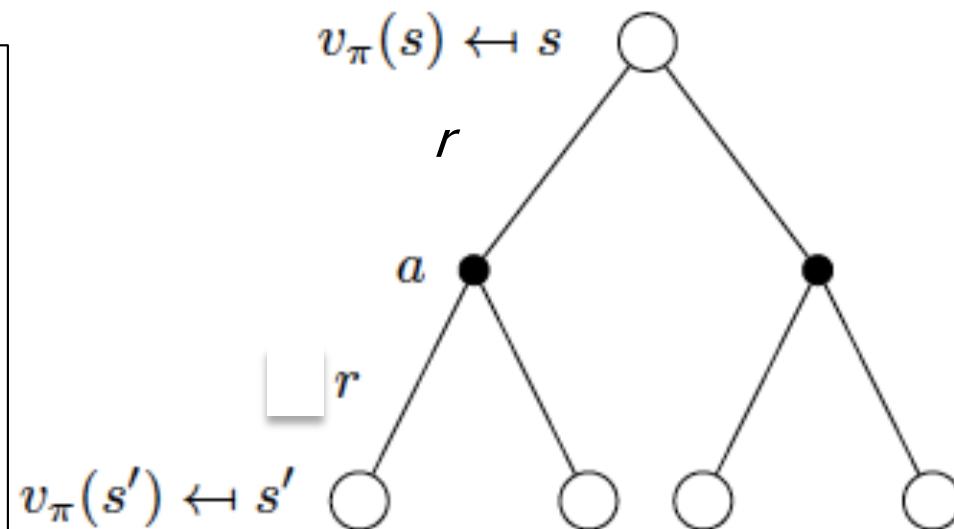
Bellman Expectation Equation for

v_π

Principle of Optimality

Decompose the problem into:

- (1) A smaller problem that is easy to solve, and
- (2) A bigger problem, that is assumed to be solved
- (3) Put the 2 parts together to solve original problem



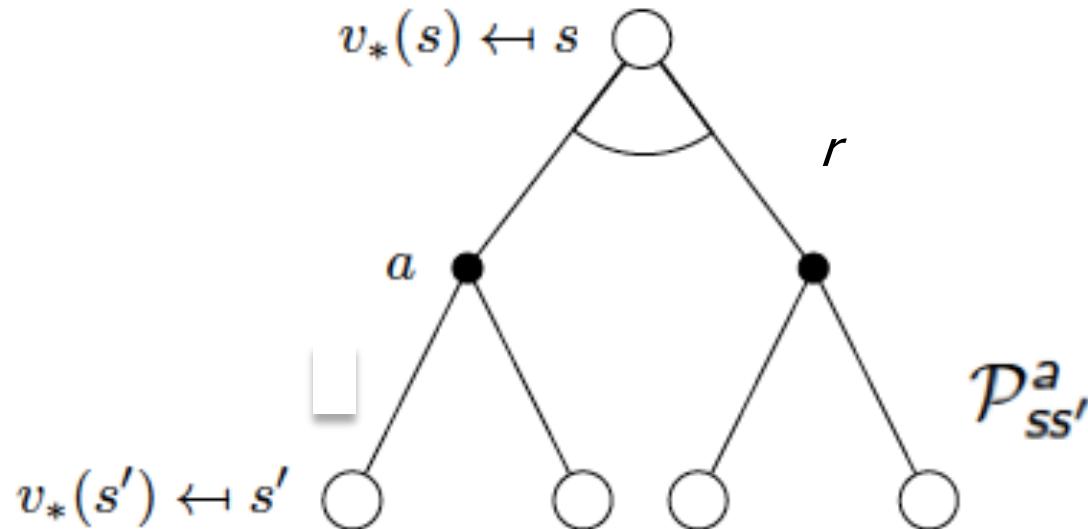
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Value Function for State s = One Step Reward + Value Function for Next State s'

If $v_{\pi_1}(s) \geq v_{\pi_2}(s)$ for all s , then $\pi_1 \geq \pi_2$

Bellman Optimality Equation for

v_*



$$v_*(s) = \max_a (\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_*(s'))$$

The Policy π_* corresponding to v_* is the Optimal Policy

Finding the Optimal Policy

$$v_*(s) = \max_a (\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s'))$$

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

$$\pi_* = \text{argmax}_a (q_*(s, a))$$

Today's Lecture: Solving the Planning Problem

- ▶ Last Lecture: We set up equations for MDPs but did not show how to solve them
- ▶ Today's Lecture: Solution Methods – Dynamic Programming
- ▶ These methods have limited utility in RL
 - They assume a perfect model
 - Computational expense
- ▶ However they are important theoretically since they provide a foundation for RL methods in rest of course

Rest of Course: Turn these methods onto scalable RL Algorithms



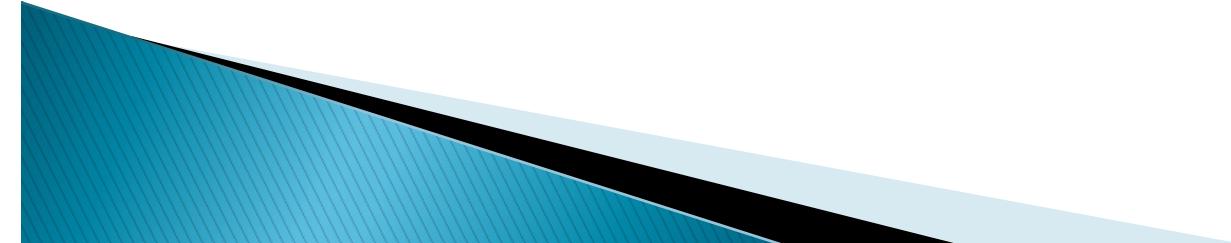
Today's Lecture: Solving the Planning Problem

We will discuss two types of Planning Algorithms:

1. Policy Evaluation: Given an MDP and a Policy, find the Value Function $v(S)$
2. Optimal Control: Given an MDP, find the Optimal Policy $\pi(S)$



Policy Evaluation



Policy Evaluation

MDP known

Policy known

Question: What are the Value Functions

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

$$q_{\pi}(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_{\pi}(s', a')$$

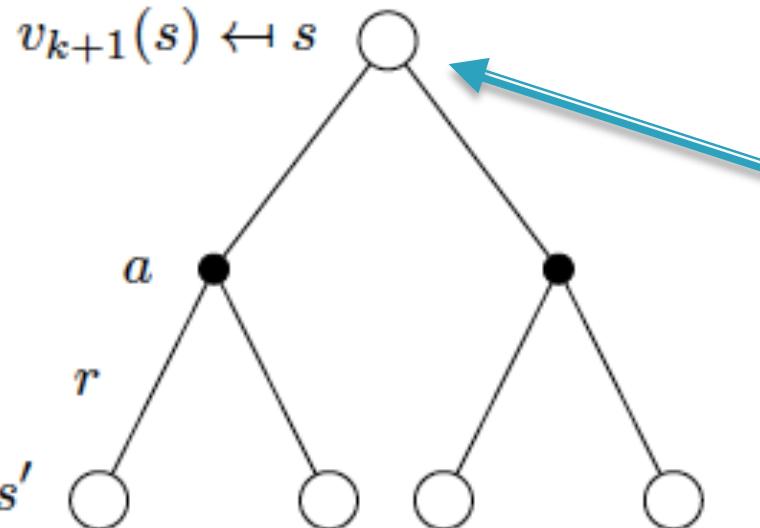
How to Solve:

- Matrix Inversion
- Iterative Methods

Iterative Policy Evaluation

Full Sweep

Full Backup



Every state gets
its turn being the
root

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$
$$\mathbf{v}^{k+1} = \mathbf{\mathcal{R}}^\pi + \gamma \mathbf{\mathcal{P}}^\pi \mathbf{v}^k$$

Bellman
Expectation
Equation

$v_{k+1}(s)$: Value function at the next iteration

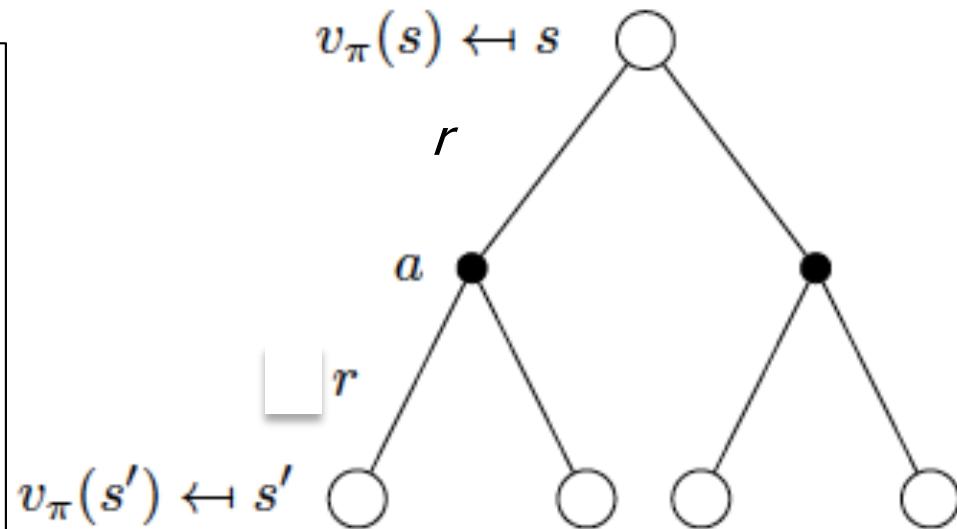
$v_k(s)$: Value function at the previous iteration

Bellman Expectation Equation for v_π

Principle of Optimality

Decompose the problem into:

- (1) A smaller problem that is easy to solve, and
- (2) A bigger problem, that is assumed to be solved
- (3) Put the 2 parts together to solve original problem



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

Value Function for State s = One Step Reward + Value Function for Next State s'

Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_\pi$
- 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

Iterative policy evaluation

Input π , the policy to be evaluated

Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

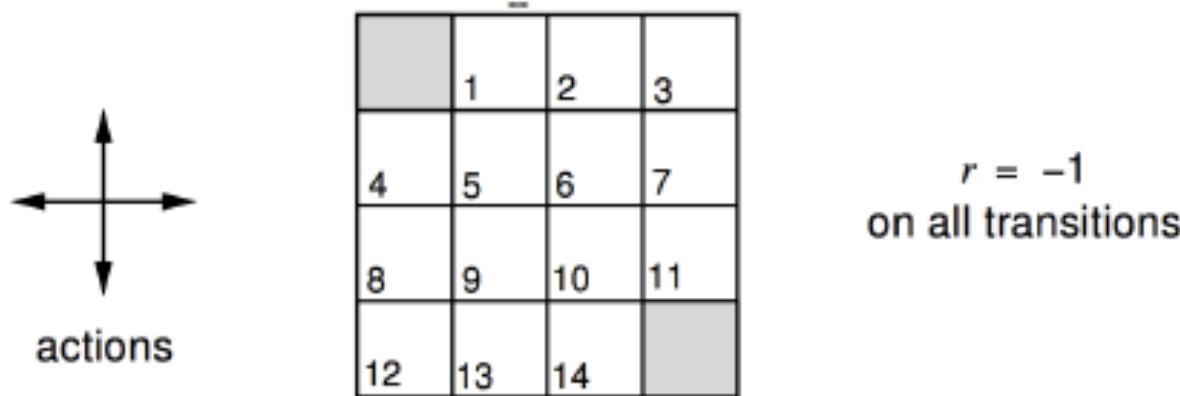
$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

Evaluating a Random Policy in a Small Gridworld



- One terminal state (shown twice as shaded squares)
- Agent follows uniform random policy

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

Iterative Policy Evaluation in a Small Gridworld

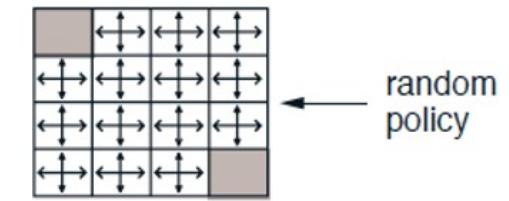
v_k for the
Random Policy

$k = 0$

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

$k = 3$

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



$k = 1$

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

$k = 10$

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

$k = 2$

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

$k = \infty$

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

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$$

Policy Iteration



Policy Iteration

Basic Idea: It is possible to find a Better Policy , while following another Policy

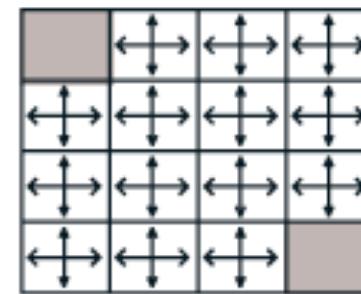


Finding the Optimal Policy

v_k for the
Random Policy

$k = 0$

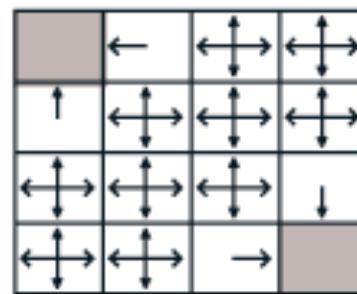
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random
policy

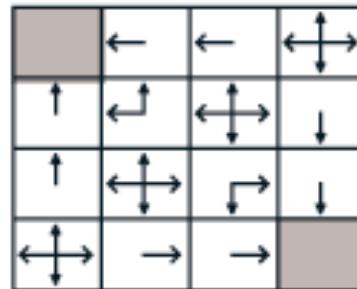
$k = 1$

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



Finding the Optimal Policy (2)

$k = 3$

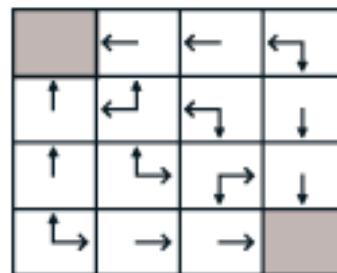
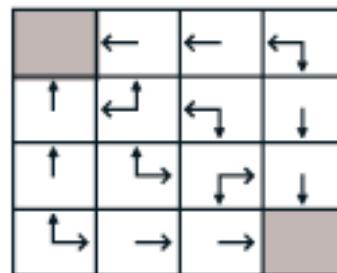
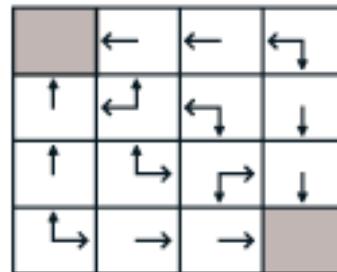
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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



optimal policy

Greedy Policy
w.r.t. v_k

$$\pi_* = \operatorname{argmax}_a(q_*(s, a))$$

We evaluated a Random Policy π_1 , but at the same time we were able to compute the Optimal Policy π_2

Improving Policies

- Given a policy π
 - Evaluate the policy π

$$v_\pi(s) = \mathbb{E} [R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

- Improve the policy by acting greedily with respect to v_π

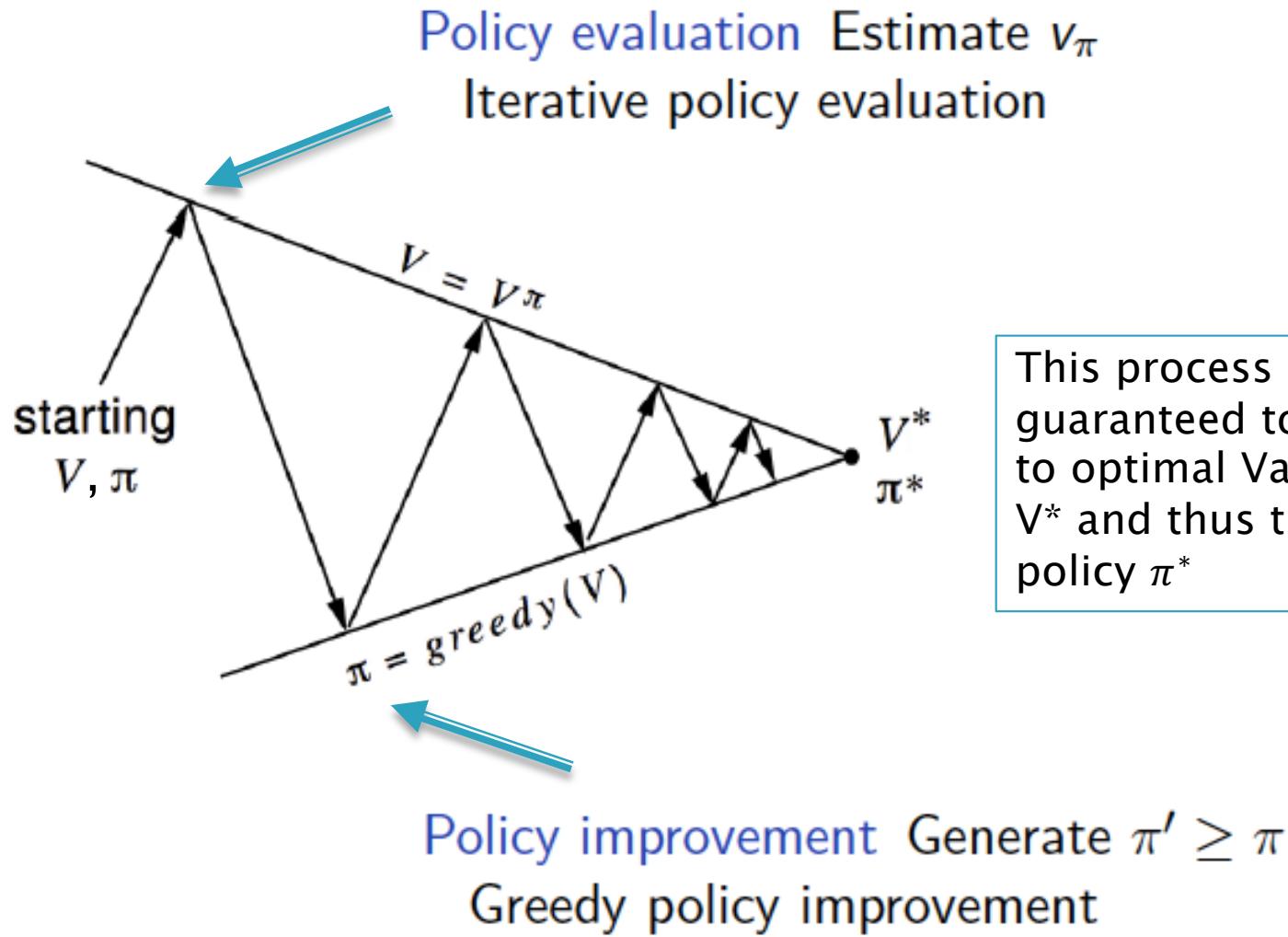
$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} q_\pi(s, a)$$



- 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^*

Compute $q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$

Policy Iteration



Policy Iteration Algorithm

Policy iteration (using iterative policy evaluation)

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

old-action $\leftarrow \pi(s)$

$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

If *old-action* $\neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Proof: Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} q_\pi(s, a)$$

- This improves the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'} [R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s] = v_{\pi'}(s) \end{aligned}$$

Policy Improvement (2)

- If improvements stop,

$$q_{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_{\pi}(s, a) = q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- Then the Bellman optimality equation has been satisfied

$$v_{\pi}(s) = \max_{a \in \mathcal{A}} q_{\pi}(s, a)$$

- Therefore $v_{\pi}(s) = v_*(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

Generalized Policy Iteration

Policy iteration (using iterative policy evaluation)

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

How many times do we need
To iterate before going on to
The Policy Improvement step?

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

$$\text{old-action} \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Generalized Policy Iteration

- Does policy evaluation need to converge to v_π ?

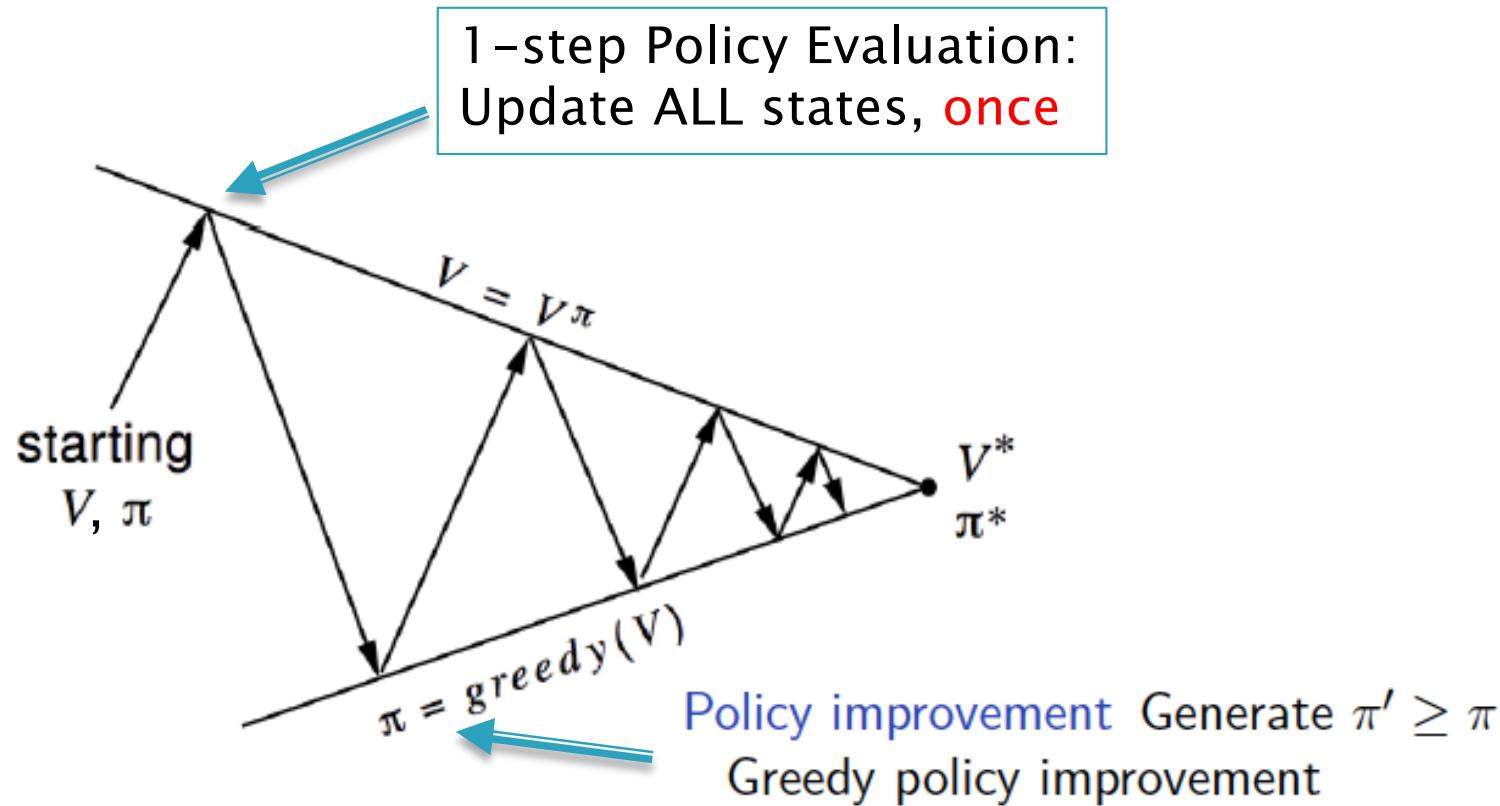
Do we need to iterate to $k = \text{infinity}$

- Why not update policy every iteration? i.e. stop after $k = 1$

This is equivalent to Value Iteration

Use approximate policy evaluation rather than exact policy evaluation

Generalized Policy Iteration (GPI)



Policy evaluation Estimate v_π

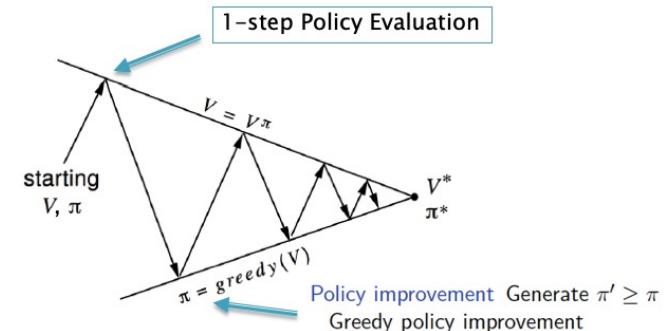
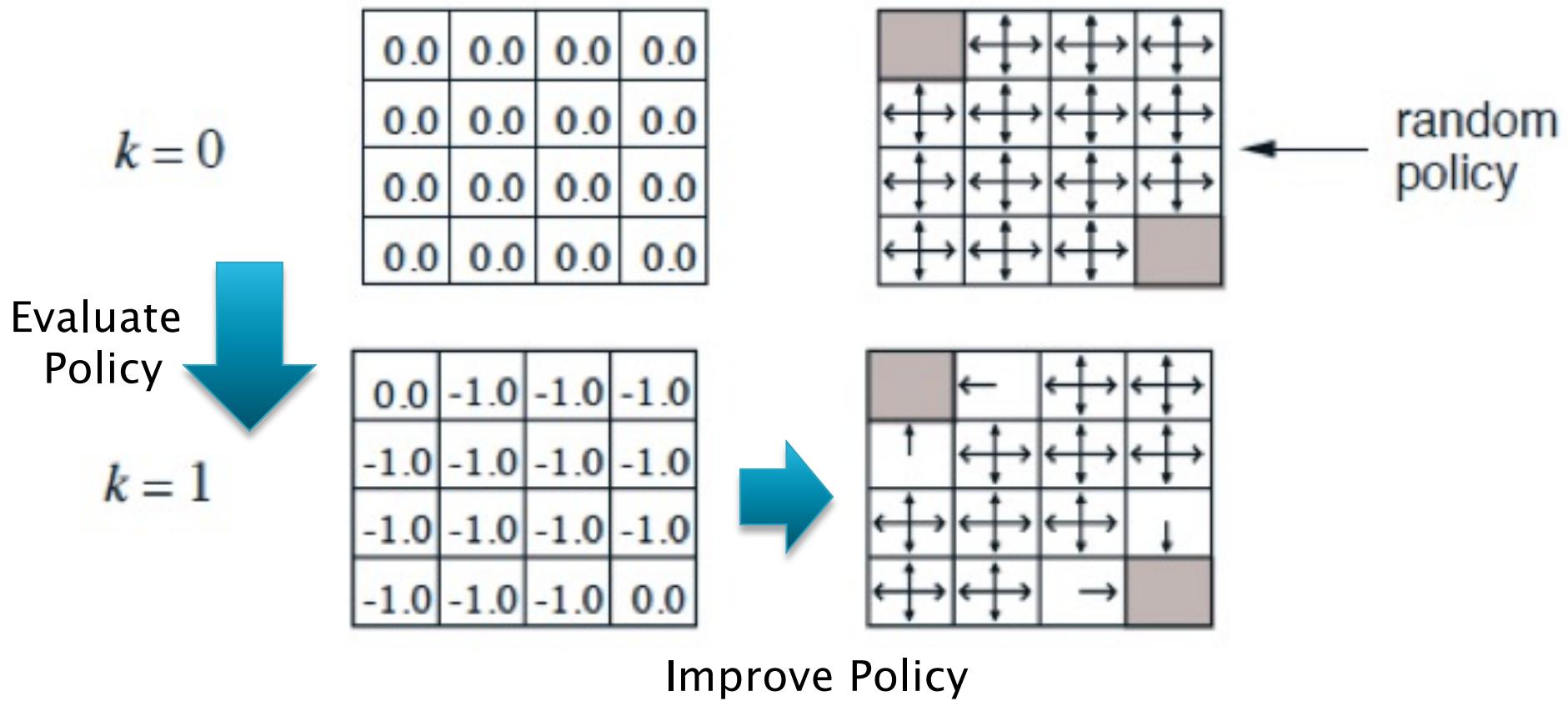
Any policy evaluation algorithm

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

Any policy improvement algorithm

Find Heuristics to be able
to solve Problems with huge
number of states and/or actions

GPI with $k = 1$



Generalized Policy Iteration

1. Iterate only for 1-step ($k = 1$), and then improve policy

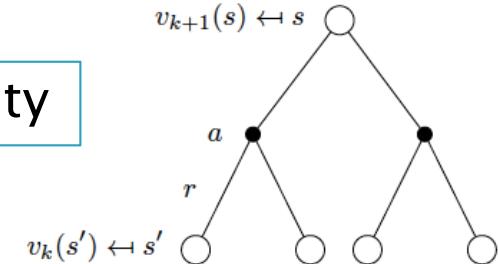


Called Value Iteration

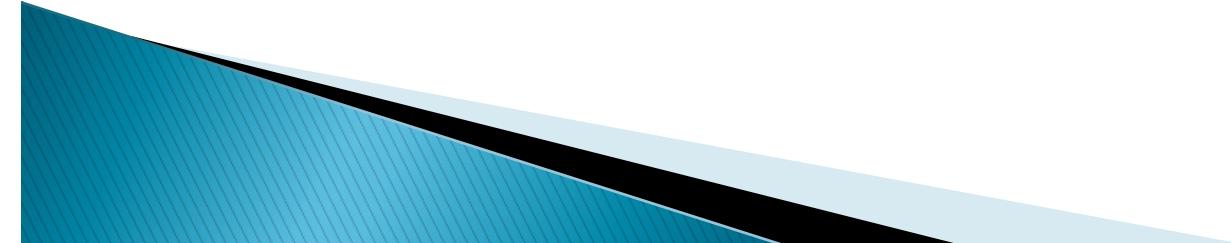
Following Lectures:

1. Iterate only some of the states, and then improve policy (even just one state!)
 - Don't do a Full Sweep!
2. Update Value Function using only a single successor state
 - Don't do a Full Backup!

Overcoming the Curse of Dimensionality

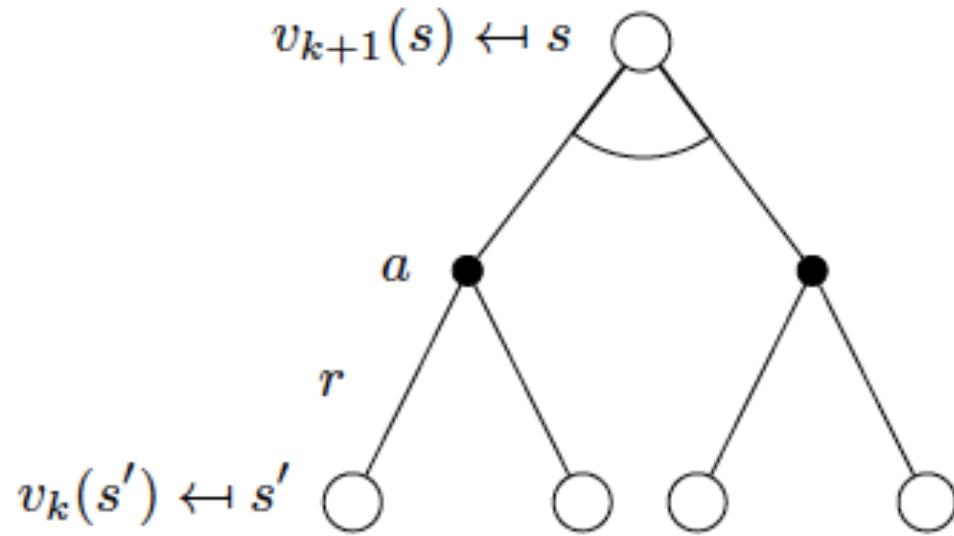


Value Iteration



Value Iteration

$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s') \right)$$



Full Sweep

Full Backup

Turn the Bellman
Optimality Equation
into an Iterative
Update

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_k(s') \right)$$

Value Iteration

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

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards



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

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- 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 Algorithm

Value iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

Contrasting Policy Iteration with Value Iteration

- ▶ In Policy Iteration we used the Bellman Expectation equation to find the Value Function for a given policy, and then iterate to find the optimal policy.
 - We alternate between Value Functions and Policies
- ▶ In Value Iteration: We take the Bellman Optimality Equation and iterate, which gives us the optimal Value Function
 - We go directly from Value Function to Value Function, there is no explicit policy

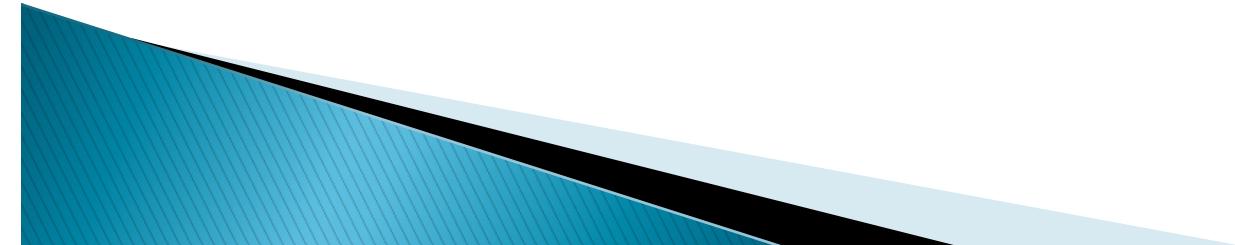


So Far: Synchronous Dynamic Programming Algorithms

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

Asynchronous Dynamic Programming



In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function
for all s in \mathcal{S}

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

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

- In-place value iteration only stores one copy of value function
for all s in \mathcal{S}

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

Plug in the latest value

Incorporates the latest information hence can be much more efficient

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

Different ways to choose which states to update:

- Real Time Dynamic Programming
- Prioritized Sweeping

Moving away from Full Sweep technique



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

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

Instead of naively updating every state, run the agent in the real World and select the states that the agent actually visits

Prioritized Sweeping

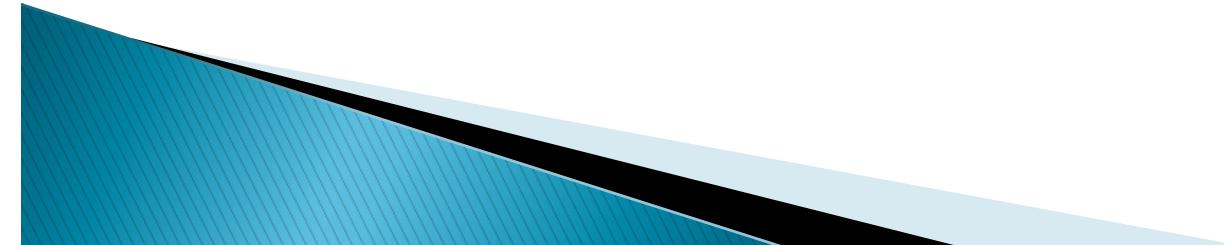
- Use magnitude of Bellman error to guide state selection, e.g.

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

- 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

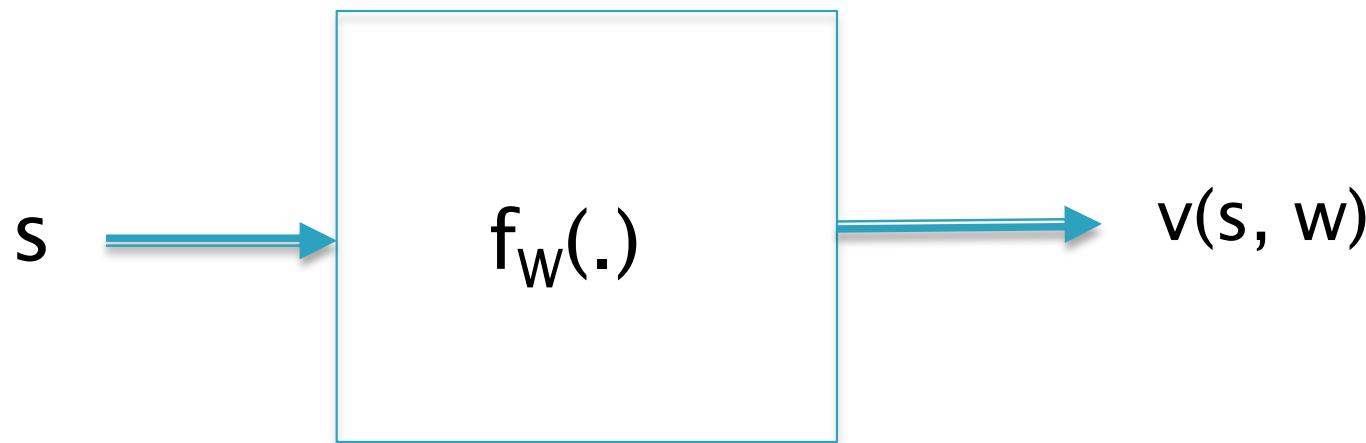
Instead of naively updating every state, select the states whose Value Functions are changing the most, ignore static states

Model Free and Approximate Dynamic Programming



Idea 1: Approximate Dynamic Programming

Function Approximator



The Dynamic Programming Equations are used to update the weights in the Function Approximator

Basic Idea: Don't update all the States. The Value Function for the non-updated states can be approximated using a Function Approximator

Approximate Dynamic Programming

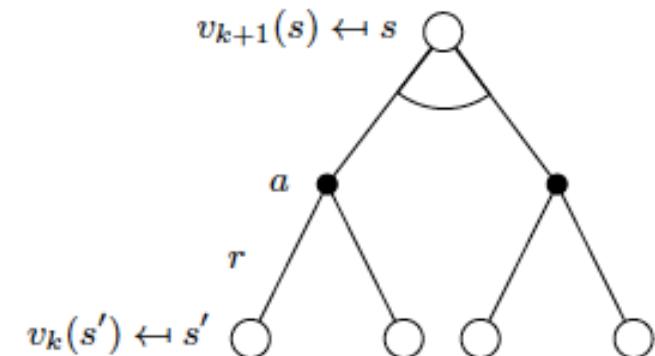
- Approximate the value function
- Using a *function approximator* $\hat{v}(s, \mathbf{w})$
- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$
- e.g. Fitted Value Iteration repeats at each iteration k ,
 - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
 - For each state $s \in \tilde{\mathcal{S}}$,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

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 $n = |S|$ grows exponentially with number of state variables
- Even one backup can be too expensive



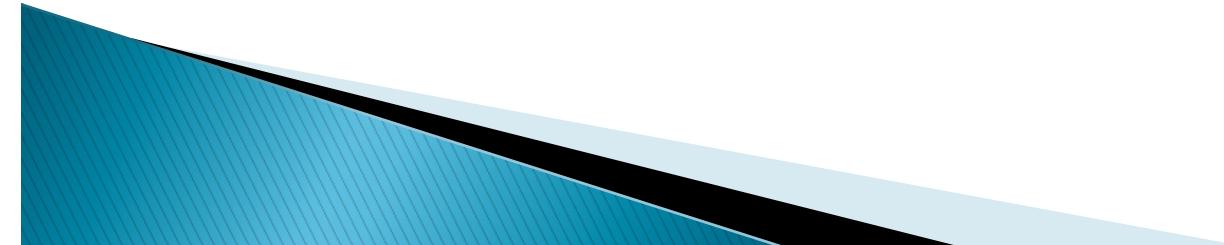
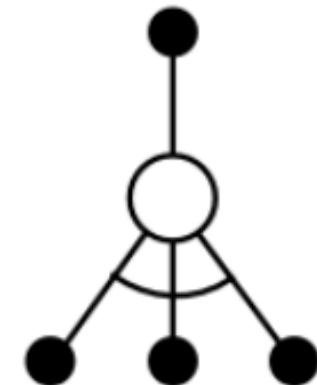
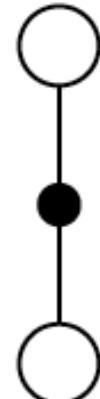
Example: An Atari screen with 170 pixels has 10^{170} states!

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

This does not scale

Idea 2: Sample Backups

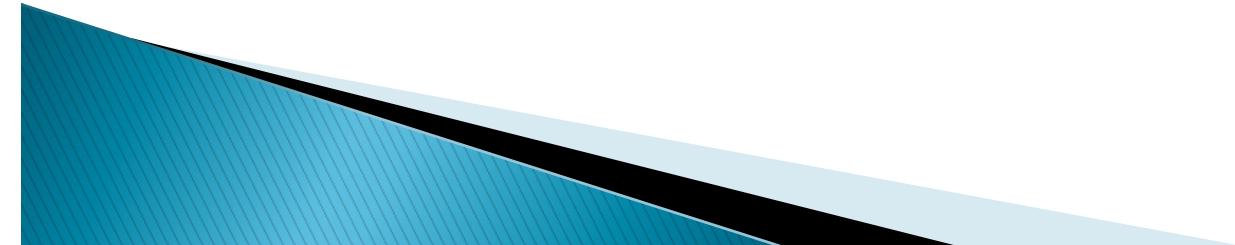
- In subsequent lectures we will consider *sample backups*
- Using sample rewards and sample transitions
 $\langle S, A, R, S' \rangle$
- Instead of reward function \mathcal{R} and transition dynamics \mathcal{P}
- Advantages:
 - Model-free: no advance knowledge of MDP required
 - Breaks the curse of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



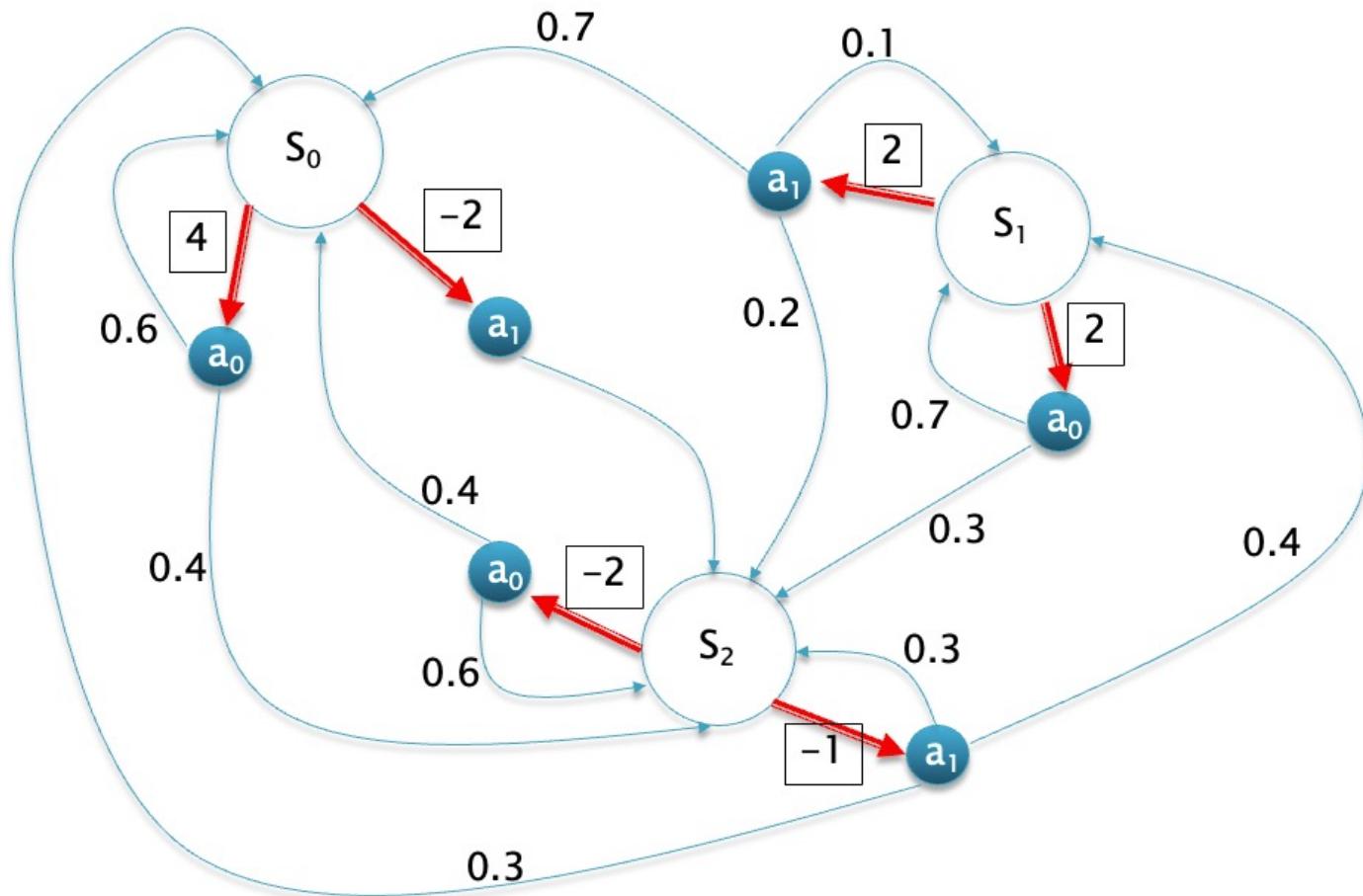
Further Reading

Sutton and Barto:

- Chapter 4



Example



Bellman Expectation Equations

$$v(0) = 0.5[4 + 0.6v(0) + 0.4v(2)] + 0.5[-2 + v(2)]$$

which results in:

$$v(0) = 1 + 0.3v(0) + 0.7v(2)$$

The equations for $v(1)$ and $v(2)$ can be similarly derived:

$$v(1) = 2 + 0.35v(0) + 0.2v(1) + 0.25v(2)$$

$$v(2) = -1.5 + 0.4v(0) + 0.2v(1) + 0.4v(2)$$



The Action Value Function

The action value function can be obtained by using the equation

$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a)v_{\pi}(s')$$

For example

$$q(0,0) = 4 + 0.6v(0) + 0.4v(2) = -1.212$$

Similarly:

$$q(0,1) = -8.07$$

$$q(1,0) = -0.82$$

$$q(1,1) = -2.6$$

$$q(2,0) = -7.45$$

$$q(2,1) = -4.78$$

