Lecture 5/6: Value and Policy Iteration

Zack Khan and Kevin Chen

Outline

Review of Last Lecture

Value Function

Definition

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

$$v_\pi(s) = \mathbb{E}_\pi [G_t \mid S_t = s]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

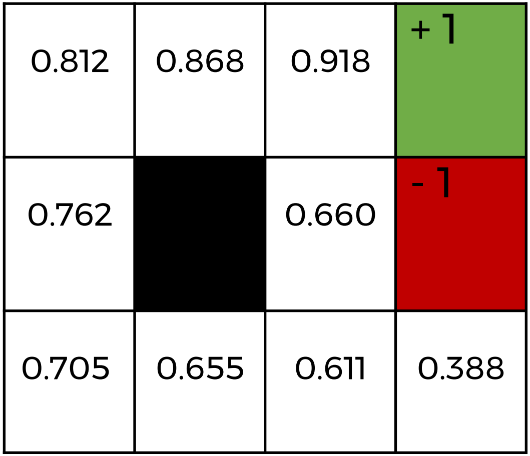
$$q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

Example State Value Function

Definition

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

$$v_\pi(s) = \mathrm{E}_\pi[G_t \mid S_t = s]$$



0.812	0.868	0.918	+ 1
0.762	■	0.660	- 1
0.705	0.655	0.611	0.388

Bellman Equation

Bellman Equation

The state-value function can be decomposed into immediate reward plus discounted value of successor state (state you end up in next)

$$v_\pi(s) = \mathrm{E}_\pi [R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s]$$

immediate Future value

The action-value function can similarly be decomposed,

$$q_\pi(s, a) = \mathrm{E}_\pi [R_{t+1} + \gamma\, q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

immediate Future value

Note how they are defined **recursively**

Bellman Expectation Equation

Value functions can be decomposed into immediate reward plus discounted value of successor states

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

immediate

Future value

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

Optimality

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.

Optimal Policy: How do we compare policies?

Define a partial ordering over policies:

A policy A is more optimal (or equally optimal) to another policy B when the value expected from following policy A is greater than or equal to the value expected than following policy B at *every state* in the MDP.

$$\pi \geq \pi \text{ if } v_\pi(s) \geq v_{\pi^j}(s), \forall s$$

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, $v_{\pi_*}(s) = v_*(s)$*
- *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 $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ q_*(s,a) \\ 0 & \textit{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Recall: Bellman Expectation Equation

Bellman Expectation Equations:

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

How do we get the highest possible value? (aka the optimal value function)?

Recall: Bellman Equation Summary

Bellman Expectation Equations:

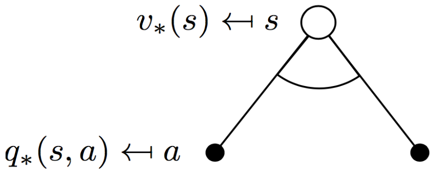
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

How do we get the highest possible value? (aka the optimal value function)?

We take the maximum action value, Q*(s,a)

Optimal Value Function for State Value

The optimal value functions are recursively related by the Bellman optimality equations:



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

Optimal State Value Function

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

Optimal State Value Function

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

Recall:

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

Optimal State Value Function

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

Recall:

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

So how do we get q*(s,a)?

Optimal State Value Function

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

Recall:

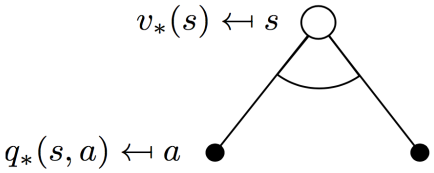
$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

So how do we get q*(s,a)? Use the optimal state value function!

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

Optimal Value Function for State Value

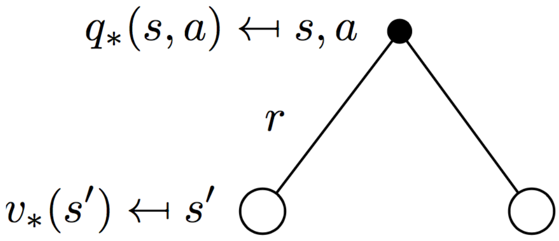
The optimal value functions are recursively related by the Bellman optimality equations:



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

Remember: we already defined q in terms of v!

Optimal Value Function for Action Value



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

Optimal Q in terms of V

Optimal State Value Function

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

Optimal State Value Function

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

Replace this with:

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

Optimal State Value Function

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

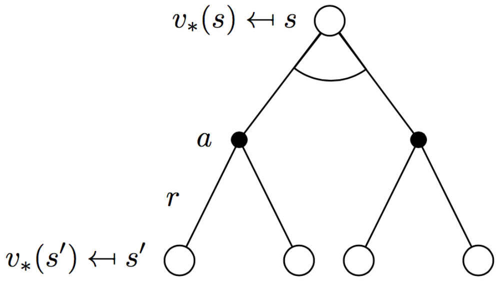
Replace this with:

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

Finally, we get:

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

Optimal Value Function for State Value (2)



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

Optimal State Value written in terms of itself!

Bellman Equation Summary

Bellman Expectation Equations:

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

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

Bellman Optimality Equations:

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

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

Dynamic Programming Methods

Introduction to Approximating Value Functions

In practice, computing value functions can be infeasible, so we will spend much of the course exploring ways to **approximate** the value function.

Our first approach will be Dynamic Programming methods. The first Dynamic Programming method we will learn to approximate the value function is **Value Iteration.**

Dynamic Programming

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

- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused
- Markov decision processes satisfy both properties
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

Value Iteration

Value Iteration Steps

Steps

- Start with Random initial values (ex: V(s) = 0)
- For each state S, calculate its new V based on its neighbor's values

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

- This is called a value update or Bellman update
- Repeat until convergence (can also stop when no value changes more than a preset parameter, δ)

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

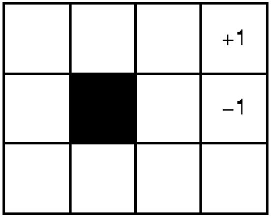
Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Steps for Value Iteration.
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$ using optimality equation

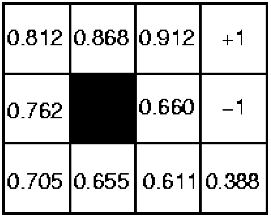
- Intermediate value functions may not correspond to any policy

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

Value Iteration and Optimal Policy

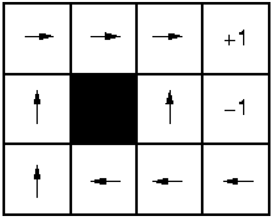


1. Given environment

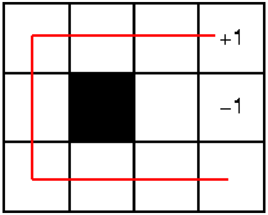


0.812	0.868	0.912	+1
0.762	■	0.660	−1
0.705	0.655	0.611	0.388

2. Calculate state values

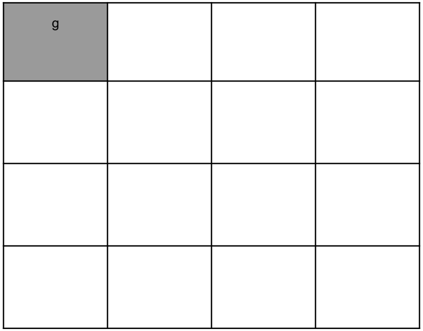


3. Extract optimal policy



4. Execute actions

Value Iteration Example



Reward of -1 for every action, only 1 goal state with reward 0 (terminal). Actions are moving up, down, left, right. If you move in a direction, you have a 100% chance of moving in that direction. Gamma = 1

Value Iteration Example

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

Initialize to random values
(in this case all 0)

Value Iteration Example Iteration 1

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

Apply Bellman Optimality Equation to every cell

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

Value Iteration Example Iteration 1

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

-1 + (1 * 0) = -1

Apply Bellman Optimality Equation to every cell

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

Value Iteration Example Iteration 1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

v_1

Apply Bellman Optimality Equation to every cell

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

Value Iteration Example Iteration 2

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

-1 + (1 * 0) = -1

-1 + (1 * -1) = -2

Apply Bellman Optimality Equation to every cell

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

Value Iteration Example Iteration 3

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

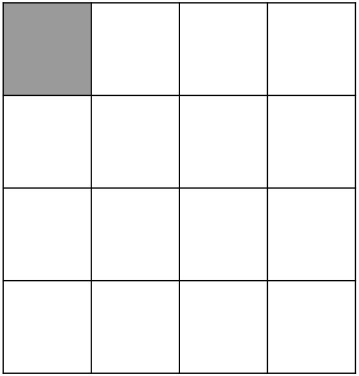
-1 + (1 * 0) = -1

-1 + (1* -1) = -2

-1 + (1 * -2) = -3

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

Value Iteration Example Iteration 4?



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

Value Iteration Example Iteration 4

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

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

Value Iteration Example

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

After Iteration 7:

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

Policy:

Greedily pick the state with highest state value

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Steps for Value Iteration.
  - At each iteration $k+1$
  - For all states $s \in \mathbb{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$ using optimality equation
- Intermediate value functions may not correspond to any policy

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

Use of Value Iteration

We can actually solve an MDP now!

Given full information about the MDP that describes our RL problem, we can now find an optimal policy.

But can we do it more efficiently?

Limitations of Value Iteration

Two weaknesses:

- Long time to converge without policy changing
- If all we really care about is the optimal policy, why not just find that policy directly?

Policy Iteration

Policy Iteration

Finding policy directly = Policy Iteration!

1. start with a random policy,
2. compute each state’s value given that policy,
3. select a new optimal policy by acting greedy on those new state values

Policy Iteration Steps

Create a random policy by selecting a random action for each state.

While not done:

(a) Compute the value for each state given the current policy.

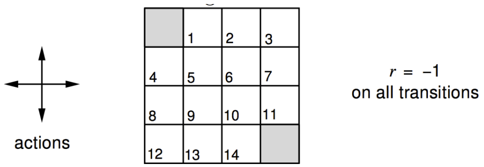
(b) Update state values using Bellman expectation equation

(c) Given these new values,
select the optimal action for each state.

If no action changes, halt

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

Policy Iteration Steps



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states 1, ..., 14
- Two terminal states (shown 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 (**probability of .25 for each action**, with action chosen randomly)

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

Policy Iteration Steps

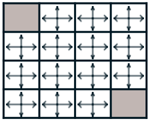
$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k=0$

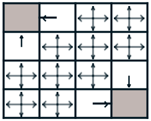
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

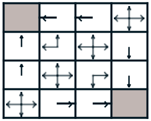
$k=1$

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 Iteration Steps

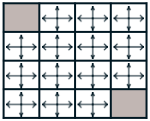
$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

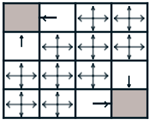
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

$k = 1$

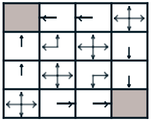
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



.25(-1 + .25*0) * 4 = -1

$k = 2$

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 Iteration Steps

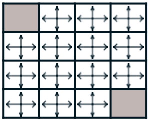
$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

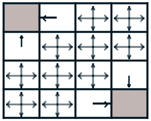
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

$k = 1$

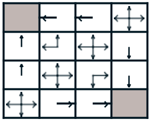
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



.25(-1 + 1*0) * 4 = -1

$k = 2$

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



.25(-1 + 1*-1) * 4 = -2

.25(-1 + 1*-1) * 3 +

.25(-1 + 1*0) *1 = -1.75

Policy Iteration Steps

$K = 2$

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

V_k for the Random Policy

Greedy Policy w.r.t. V_k

$k = 3$

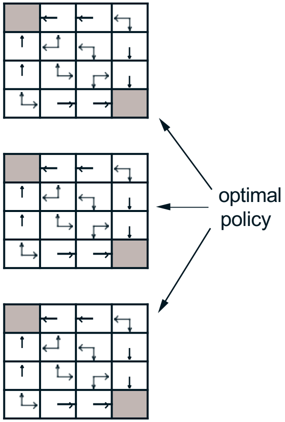
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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



Policy Iteration Steps

$K = 2$

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

$.25(-1 + 0) + .25(-1 + -1.75) +$
$.25(-1 + -2) * 2 =$ 2.4

$.25(-1 + -2) *4 =$ -3.0

$.25(-1 + -1.75) +$
$.25(-1 + -2) * 3 =$ -2.9

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 3$

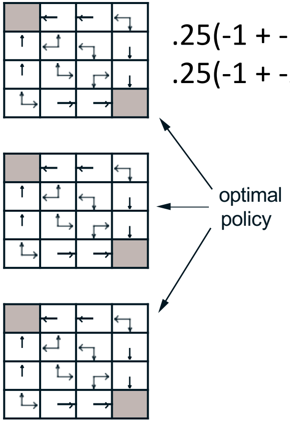
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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



Policy Iteration Steps

- Given a policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathrm{E}\left[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s\right]$$

  - 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 $\pi*$

(We always converge to the optimal policy!)

Policy Iteration Summary

Another way of solving MDP, like Value Iteration, but directly solves for policy.

Usually requires less iterations, and uses Bellman Expectation Equation (Value Iteration uses Bellman Optimality Equation)

Next Week: Model-Free Methods

What if we didn't have all this information about the MDP?

Next Week: Model-Free Methods

What if we didn’t have all this information about the MDP?

Model Free Methods! (Next Lecture!)

Questions?