# Morkov Decision Processes

# Overview

- Decision processes and Markov Decision Processes (MDP)
- Rewards and Optimal Policies
- Defining features of Markov Decision Process
- Solving MDPs
  - Value Iteration
  - Policy Iteration
- POMDPs

# Rewiew: Decisions Under Uncertainty

- Some areas of AI (e.g., planning) focus on decision making in domains where the environment is understood with certainty
- Here we focus on an agent that needs to make decisions in a domain that involves uncertainty
- An agent's decision will depend on:
  - what actions are available. They often don't have deterministic outcome
  - what beliefs the agent has over the world
  - the agent's goals and preferences

برای این که فرایند تصمیم گیری داشته باشیم
باید یک عدم قطعیت در محیط باشد

# Decision Processes

- We focus on situations that involve sequences of decisions
  - The agent decides which action to perform
  - The new state of the world depends probabilistically upon the previous state as well as the action performed
  - The agent receives rewards or punishments at various points in the process
  - The agent's utility depends upon the final state reached, and the sequence of actions taken to get there
- Aim: maximize the reward received

دنباله ای از decision ها می گویند
در هر لحظه کاری انجام می دهیم و نتیجه ای از آن می گیریم
و هدف در نهایت این است که بیشترین reward را به دست آوریم

# Markov Decision Processes (MDP)

- For an MDP you specify:
  - set S of states, set A of actions
  - Initial state $s_0$
  - the process' dynamics (or ***transition model***)

    $P(s'|s,a)$

    اگر تابع توزیع احتمال فقط به state فعلی و کنشی که انجام می دهیم بستگی داشته باشد خاصیت Markov دارد
    یعنی هیچ ارتباطی با نحوه رسیدن به این state ندارد .
    بنابراین فقط مربوط به state فعلی و کنشی که می خواهیم انجام بدهیم باشد .
    در markov به ازای هر state یک reward می گیریم

    اگر در خانه s باشیم و کنش aرا انجام دهیم به خانه s prime می رویم
    یک تابع توزیع احتمال است
  - The reward function

    $R(s, a, s')$

    describing the reward that the agent receives when it performs action *a* in state *s* and ends up in state *s'*
  - We will use R(s) when the reward depends only on the state s and not on how the agent got there

# Planning Horizons

- The ***planning horizon*** defines the timing for decision making
  - ***Indefinite horizons***: the decision process ends but it is unknown when
    - ✓ In the previous example, the process ends when the agent enters one of the two terminal (*absorbing*) states
      حالتی است که stateهایی وجود دارد که فرآیند در آن تمام می شود(جذب کننده داریم ) ولی نمی دانیم این stateها با چند کنش به دست می آیند
  - ***Infinite horizons***: the process never halts
    فرایند های markov را بی نهایت میباشد
    - ✓ e.g. if we change the previous example so that, when the agent enters one of the terminal states, it is flung back to one of the two left corners of the grid
  - ***Finite horizons***: the process must end at a specific time N
    برای حل مساله باید k کنش انجام داد و بیش از آن نیازی نیست
- We focus on MDPs with infinite and indefinite horizons

# Information Availability

- What information is available when the agent decides what to do?
- ***Fully-observable MDP (FOMDP):*** در هر لحظه state را میبنیم و می دانیم کجای MDP هستیم
  - the agent gets to observe current state $s_t$ when deciding on action $a_t$ .
- ***Partially-observable MDP (POMDP)*** نمی دانیم در کجای MDP هستیم
  - the agent can't observe $s_t$ directly, can only use sensors to get information
- We only look at FOMDP (but we will call them simply MDPs)

# Reward Function

- Suppose the agent goes through states $s_1, s_2,...,s_k$ and receives rewards $r_1, r_2,...,r_k$
- We will look at ***discounted reward*** to define the reward for this sequence, i.e. its *utility* for the agent

$$\gamma \ \textbf{\textit{discount factor}}, \ 0 \le \gamma \le 1$$

ضریب تخفیف : در هر لحظه ارزش rewardهایی که دیرتر می گیریم را کمتر می کند

$R_{\max}$ bound on R(s) for every $s$

$$U[s_1, s_2, s_3, ..] = r_1 + \gamma r_2 + \gamma^2 r_3 + .....$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{i+1} \le \sum_{i=0}^{\infty} \gamma^i R_{\max} = \frac{R_{\max}}{1-\gamma}$$

اگر پاداش بی نهایت نداشته باشیم بنابراین مجموع reward ها تا بی نهایت متناهی می شود (مجموع تخفیف داده شده )
به این دلیل که تبدیل به یک تصاعد هندسی می شود با قدر نسبت gama

# Solving MDPs

- In search problems, aim is to find an optimal *state sequence*

به جای این که مسیر بهینه پیدا کنیم یک mapping بین state ها و کنش ها به دست می آوریم
که به این mapping میگویند policy

- In MDPs, aim is to find an optimal *policy* π(s)
  - A policy π(s) specifies what the agent should do for each state s
  - Because of the stochastic nature of the environment, a policy can generate a set of environment histories (sequences of states) with different probabilities
- Optimal policy maximizes the *expected total reward,* where the expectation is take over the set of possible state sequences generated by the policy
  - Each state sequence associated with that policy has a given amount of total reward
  - Total reward is a function of the rewards of its individual states (we'll see how)

# More formally

- A stationary policy is a function: $\pi : S \rightarrow A$
  - For every state s, *л(s)* specifies which action should be taken in that state.
- Utility of a state
  - Defined as the expected utility of the state sequences that may follow it
  - These sequences depend on the policy *л* being followed, so we define the utility of a state s with respected to *л* (a.k.a ***value*** of *л* for that state) *as*

$$U^{\pi}(s) = E\left[\sum_{i=0}^{\infty} \gamma^{i} R(S_i)\right] \quad \text{where } S_0 = s$$

امید ریاضی میزان پاداش هایی که ما از این state به بعد به دست می آوریم را نشان میدهد .

  - Where $S_i$ is the random variable representing the state the agent reaches after executing *л* for *i* steps starting from *s*

# More Formally (cont.)

- An optimal policy is one with maximum expected discounted reward for every state.
  - A policy $\pi^*$ is optimal if there is no other policy $\pi'$ and state s such as

    $U^{\pi'}(s) > U^{\pi^*}(s)$

  - That is an optimal policy gives the Maximum Expected Utility (MEU)

- For a fully-observable MDP with ***stationary dynamics*** and ***rewards*** with ***infinite*** or ***indefinite horizon***, there is always an optimal stationary policy.

# Value Iteration

- Algorithm to find an optimal policy and its value for a MDP
- The idea is to find the utilities of states, and use them to select the action with the maximum expected utility for each state

ایده الگوریتم ها برای به دست آوردن policyبهینه :
مقدار policy بهینه برای هر state را مربوط می کنند به
مقدار policy بهینه برای stateهای همسایه (stateهایی که با کنشی می توان به آن ها رفت )آن

# Value Iteration: from state utilities to $\pi^*$

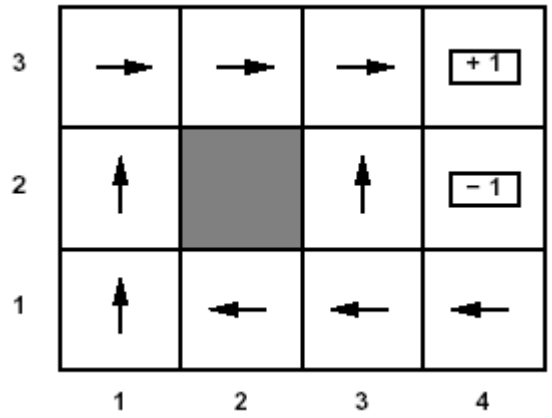
- Remember that we defined the utility of a state *s* as the expected sum of the possible discounted rewards from that point onward

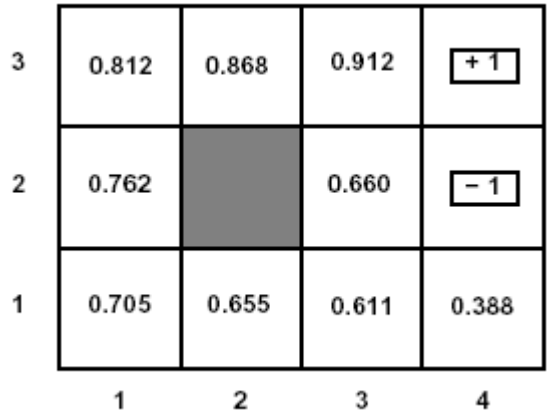
$$U^{\pi}(s) = E\left[\sum_{i=0}^{\infty} \gamma^i R(S_i)\right] \quad \text{where } S_0 = s$$

- We define *U(s)* as the utility of state s when the agent follows the optimal policy $\pi^*$ from s onward *(i. e., $U^{\pi^*}(s)$)*
  - aka value of $\pi^*$

# U(s) and R(s)

- Note that U(s) and R(s) are quite different quantities
  - R(s) is the short term reward related to s
  - U(s) is the expected long term total reward from s onward if following $\pi^*$
- Example: U(s) for the optimal policy we found for the grid problem with *R(s non-terminal)* = -0.04 and $\gamma = 1$



# Value Iteration: from state utilities to $\pi^*$

- Now the agent can chose the action that implements the MEU principle: maximize the expected utility of the subsequent state

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s, a) U(s')$$

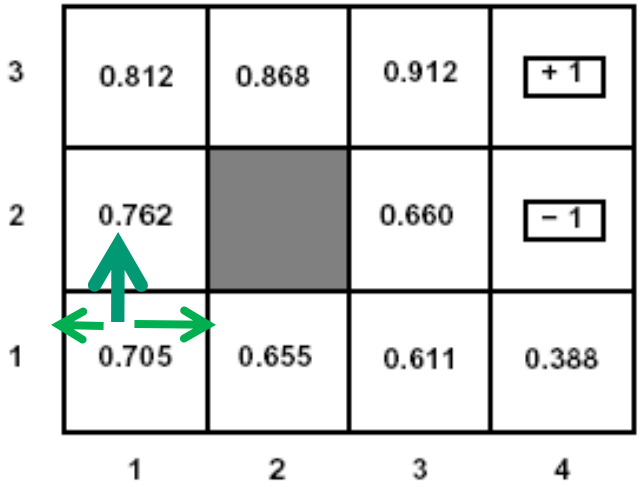
expected value of following policy л* in s'

states reachable from s by doing a

Probability of getting to *s'* from *s* via *a*

# Example

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)U(s')$$




با احتمال ۰.۱ کاری را انجام می دهد که ۹۰ درجه با کار درست فاصله دارد

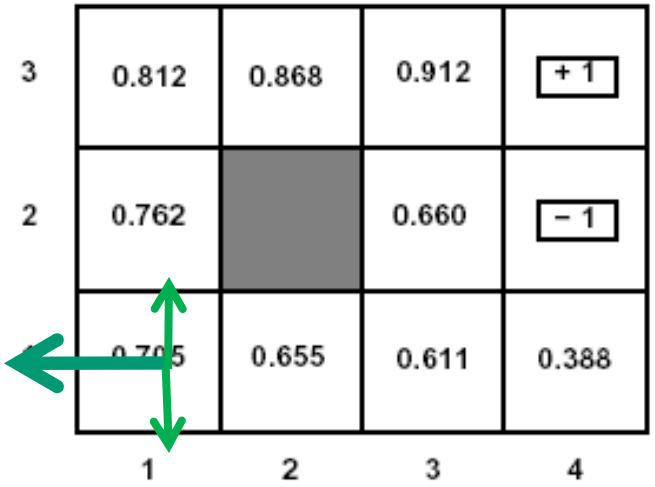
- To find the best action in (1,1)

$$\pi^*(1,1) = \arg\max \begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) & UP \\ 0.9U(1,1) + 0.1U(1,2) & LEFT \\ 0.9U(1,1) + 0.1U(2,1) & DOWN \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) & RIGHT \end{bmatrix}$$

Action

# Example

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)U(s')$$

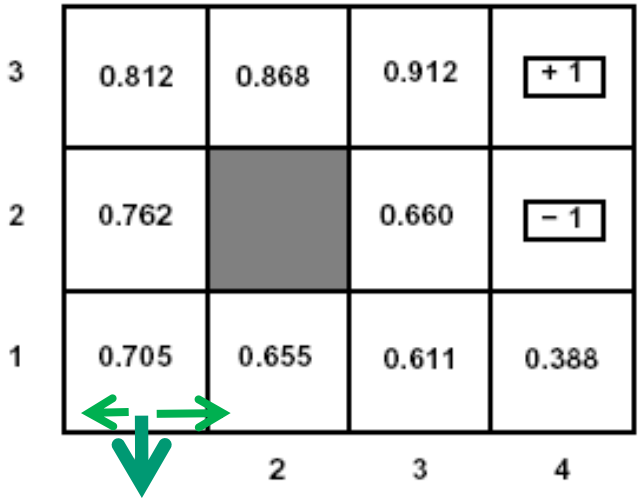



- To find the best action in (1,1)

$$\pi^*(1,1) = \arg\max \begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) & UP \\ 0.9U(1,1) + 0.1U(1,2) & LEFT \\ 0.9U(1,1) + 0.1U(2,1) & DOWN \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) & RIGHT \end{bmatrix}$$

# Example

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s, a) U(s')$$

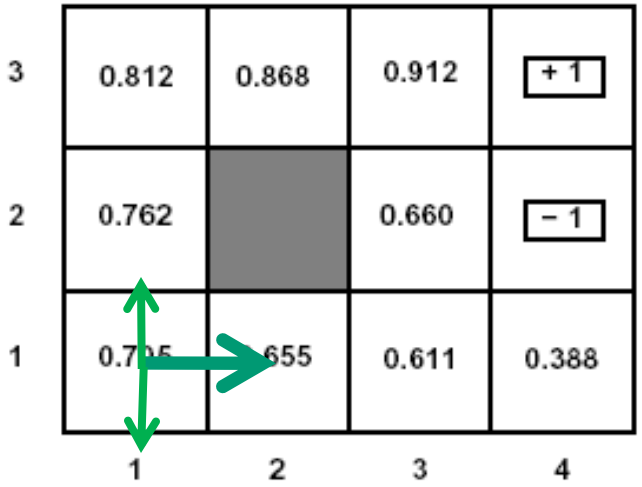



- To find the best action in (1,1)

$$\pi^*(1,1) = \arg\max \begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) & UP \\ 0.9U(1,1) + 0.1U(1,2) & LEFT \\ 0.9U(1,1) + 0.1U(2,1) & DOWN \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) & RIGHT \end{bmatrix}$$

# Example

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)U(s')$$




- To find the best action in (1,1)

$$\pi^*(1,1) = \arg\max \begin{bmatrix} 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) & UP \\ 0.9U(1,1) + 0.1U(1,2) & LEFT \\ 0.9U(1,1) + 0.1U(2,1) & DOWN \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) & RIGHT \end{bmatrix}$$

- Plugging in the numbers from the game state above, give *Up* as best action

# Finding Utilities

- Great, but how do we find the utilities?
- Value iteration exploits the relationship between the utility of a state and the utility of its neighbours

$$U(s_k) = E\left[\sum_{i=0}^{\infty}\gamma^i R(S_{k+i})\right] = E\left[R(S_k) + \sum_{i=1}^{\infty}\gamma^i R(S_{k+i})\right]$$

$$= E\left[R(s_k) + \gamma\sum_{i=1}^{\infty}\gamma^{i-1} R(S_{k+i})\right] = R(s_k) + \gamma E\left[\sum_{i=1}^{\infty}\gamma^{i-1} R(S_{k+i})\right] \quad =U(S\_(k+1))$$

Reward sequences for the states reached by following the optimal action in $s_k$

- So the utility of following the optimal policy in *s* is

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U(s')$$

Immediate reward for s

Expected utility of state s' reached by action a in s

# Finding Utilities

- Given *N* states, we can write an equation like the one below for each of them

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)U(s')$$

- Each equation contains *N* unknowns – the U values for the *N* states
- N equations in N variables (Bellman equations)
  - It can be shown that they have a unique solution: the values for the optimal policy
- Unfortunately the N equations are non-linear, because of the max operator
  - Cannot be easily solved by using techniques from linear algebra
- Value Iteration Algorithm
  - Iterative approach to find the optimal policy and corresponding values

# Value Iteration: General Idea

- Let $U^{(i)}(s)$ be the utility of state $s$ at the $i^{th}$ iteration of the algorithm
- Start with arbitrary utilities on each state $s$: $U^{(0)}(s)$
- Repeat simultaneously for every $s$ until there is "no change"

$$U^{(k+1)}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U^{(k)}(s')$$

یک حدی برای خطا می گذاریم که همگرا شدن این الگوریتم را تضمین کنیم
هر دفعه که updateرا برای stateهای مختلف انجام می دهیم بررسی می کنیم که
مقدار stateها چه قدر تغییر کرده و ماکسیمم می گیریم بین میزان تغییرات stateهای مختلف
و این مقدار باید از آن حدی که گذاشتیم کمتر شود .

- True "no change" in the values of U(s) from one iteration to the next are guaranteed only if run for infinitely long.
  - In the limit, this process converges to a unique set of solutions for the Bellman equations
  - They are the total expected rewards (utilities) for the optimal policy

# VI algorithm

**function** VALUE-ITERATION($mdp, \epsilon$) **returns** a utility function
  **inputs:** $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount $\gamma$
    $\epsilon$, the maximum error allowed in the utility of any state
  **local variables:** $U$, $U'$, vectors of utilities for states in $S$, initially zero
    $\delta$, the maximum change in the utility of any state in an iteration

  **repeat**
    $U \leftarrow U'$; $\delta \leftarrow 0$
    **for each** state $s$ **in** $S$ **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      **if** $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  **until** $\delta < \epsilon(1-\gamma)/\gamma$
  **return** $U$

# Example

- Suppose, for instance, that we start with values $U^{(0)}(s)$ that are all 0

**Iteration 0**

| | | | | |
|---|---|---|---|---|
| 3 | 0 | 0 | 0 | +1 |
| 2 | 0 | | 0 | -1 |
| 1 | 0 | 0 | 0 | 0 |
| | 1 | 2 | 3 | 4 |

**Iteration 1**

| | | | | |
|---|---|---|---|---|
| 3 | 0 | 0 | 0 | +1 |
| 2 | 0 | | 0 | -1 |
| 1 | -0.04 | 0 | 0 | 0 |
| | 1 | 2 | 3 | 4 |

$$U^{(1)}(1,1) = -0.04 + 1 * \max \begin{bmatrix} 0.8U^{(0)}(1,2) + 0.1U^{(0)}(2,1) + 0.1U^{(0)}(1,1) & UP \\ 0.9U^{(0)}(1,1) + 0.1U^{(0)}(1,2) & LEFT \\ 0.9U^{(0)}(1,1) + 0.1U^{(0)}(2,1) & DOWN \\ 0.8U^{(0)}(2,1) + 0.1U^{(0)}(1,2) + 0.1U^{(0)}(1,1) & RIGHT \end{bmatrix}$$

$$U^{(1)}(1,1) = -0.04 + \max \begin{bmatrix} 0 & UP \\ 0 & LEFT \\ 0 & DOWN \\ 0 & RIGHT \end{bmatrix}$$

# Example (cont'd)

- Let's compute $U^{(1)}(3,3)$

Iteration 0

| | | | | |
|---|---|---|---|---|
| 3 | 0 | 0 | 0 | +1 |
| 2 | 0 | | 0 | -1 |
| 1 | 0 | 0 | 0 | 0 |
| | 1 | 2 | 3 | 4 |

Iteration 1

| | | | | |
|---|---|---|---|---|
| 3 | 0 | 0 | 0.76 | +1 |
| 2 | 0 | | 0 | -1 |
| 1 | -0.04 | 0 | 0 | 0 |
| | 1 | 2 | 3 | 4 |

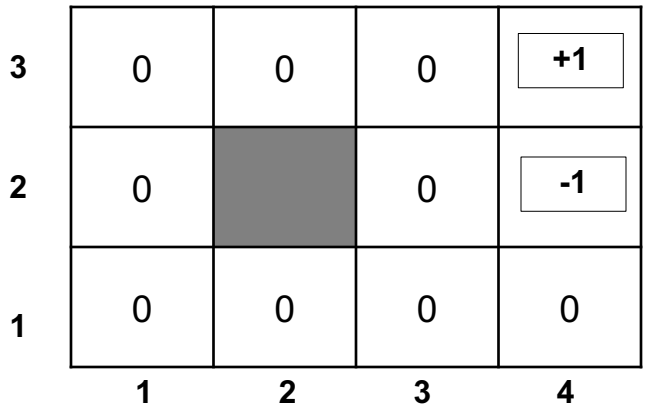
$$U^{(1)}(3,3) = -0.04 + 1 * \max \begin{bmatrix} 0.8U^{(0)}(3,3) + 0.1U^{(0)}(2,3) + 0.1U^{(0)}(4,3) & UP \\ 0.8U^{(0)}(2,3) + 0.1U^{(0)}(3,3) + 0.1U^{(0)}(3,2) & LEFT \\ 0.8U^{(0)}(3,2) + 0.1U^{(0)}(2,3) + 0.1U^{(0)}(4,3) & DOWN \\ 0.8U^{(0)}(4,3) + 0.1U^{(0)}(3,3) + 0.1U^{(0)}(3,2) & RIGHT \end{bmatrix}$$

$$U^{(1)}(3,3) = -0.04 + \max \begin{bmatrix} 0.1 & UP \\ 0 & LEFT \\ 0.1 & DOWN \\ 0.8 & RIGHT \end{bmatrix}$$

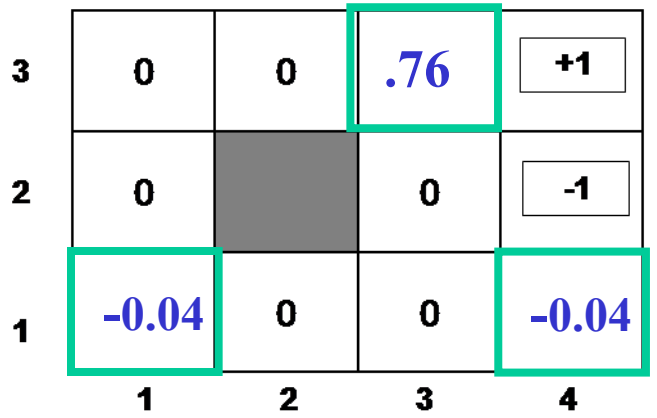
# Example (cont'd)

- Let's compute $U^{(1)}(4,1)$

Iteration 0



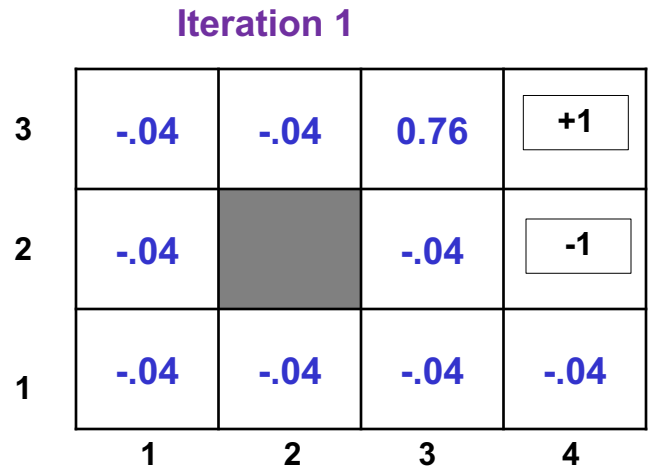

Iteration 1




$$U^{(1)}(4,1) = -0.04 + \max \begin{bmatrix} 0.8U^{(0)}(4,2) + 0.1U^{(0)}(3,1) + 0.1U^{(0)}(4,1) & UP \\ 0.8U^{(0)}(3,1) + 0.1U^{(0)}(4,2) + 0.1U^{(0)}(4,1) & LEFT \\ 0.8U^{(0)}(4,1) + 0.1U^{(0)}(3,2) + 0.1U^{(0)}(4,1) & DOWN \\ 0.8U^{(0)}(4,1) + 0.1U^{(0)}(4,2) + 0.1U^{(0)}(4,1) & RIGHT \end{bmatrix}$$

$$U^{(1)}(4,1) = -0.04 + \max \begin{bmatrix} -0.8 & UP \\ -0.1 & LEFT \\ 0 & DOWN \\ -0.1 & RIGHT \end{bmatrix}$$

# After a Full Iteration




- Only the state one step away from a positive reward (3,3) has gained value, all the others are losing value because of the cost of moving

# Some steps in the second iteration

Iteration 1

| | | | | |
|---|---|---|---|---|
| 3 | -.04 | -.04 | 0.76 | +1 |
| 2 | -.04 | | -.04 | -1 |
| 1 | -.04 | -.04 | -.04 | -.04 |
| | 1 | 2 | 3 | 4 |

Iteration 2

| | | | | |
|---|---|---|---|---|
| 3 | -.04 | -.04 | 0.76 | +1 |
| 2 | -.04 | | -.04 | -1 |
| 1 | -0.08 | -.04 | -.04 | -.04 |
| | 1 | 2 | 3 | 4 |

$$U^{(2)}(1,1) = -0.04 + 1 * \max \begin{bmatrix} 0.8U^{(1)}(1,2) + 0.1U^{(1)}(2,1) + 0.1U^{(1)}(1,1) & UP \\ 0.9U^{(1)}(1,1) + 0.1U^{(1)}(1,2) & LEFT \\ 0.9U^{(1)}(1,1) + 0.1U^{(1)}(2,1) & DOWN \\ 0.8U^{(1)}(2,1) + 0.1U^{(1)}(1,2) + 0.1U^{(1)}(1,1) & RIGHT \end{bmatrix}$$

$$U^{(2)}(1,1) = -0.04 + \max \begin{bmatrix} -.04 & UP \\ -.04 & LEFT \\ -.04 & DOWN \\ -.04 & RIGHT \end{bmatrix} = -0.08$$

# Example (cont'd)

- Let's compute $U^{(1)}(2,3)$

Iteration 1

| | | | | |
|---|---|---|---|---|
| 3 | -.04 | -.04 | 0.76 | +1 |
| 2 | -.04 | | -.04 | -1 |
| 1 | -.04 | -.04 | -.04 | -.04 |
| | 1 | 2 | 3 | 4 |

Iteration 2

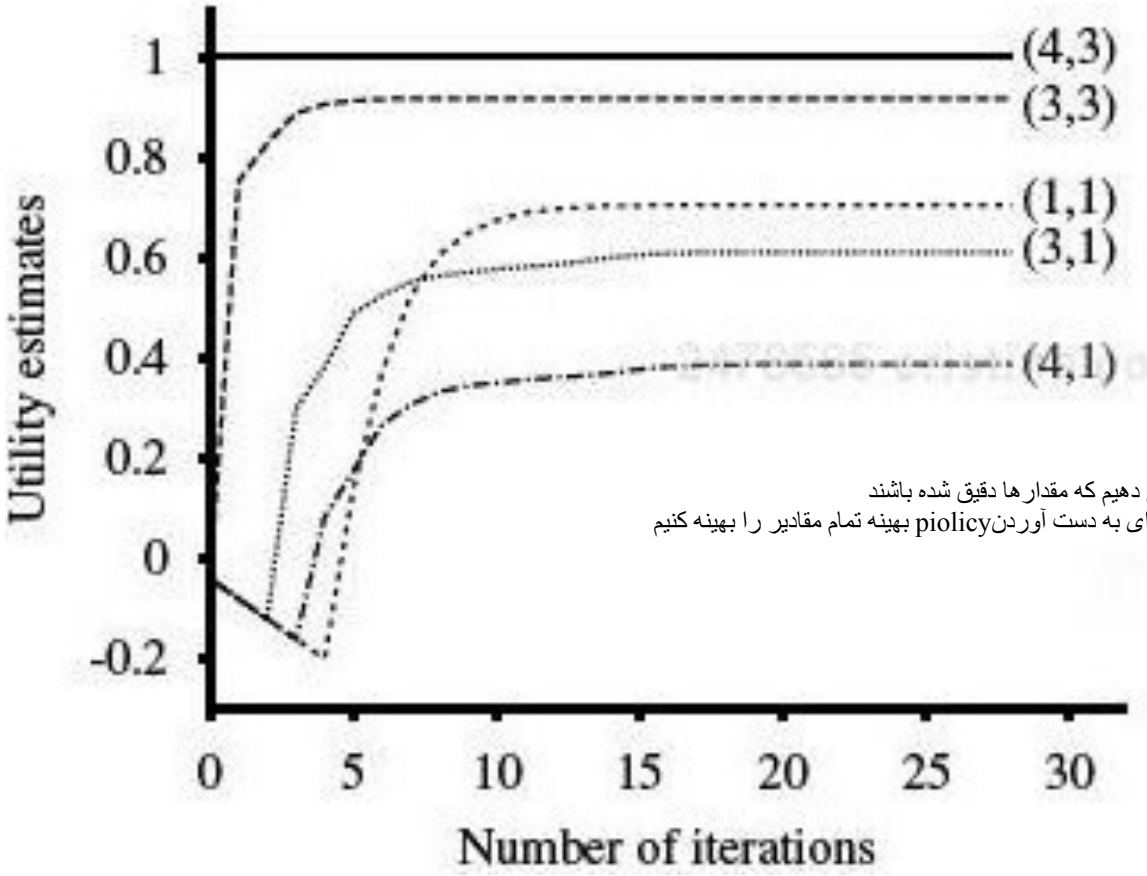
| | | | | |
|---|---|---|---|---|
| | -.04 | 0.56 | 0.76 | +1 |
| | -.04 | | -.04 | -1 |
| | -0.08 | -.04 | -.04 | -.04 |
| | 1 | 2 | 3 | 4 |

$$U^{(1)}(2,3) = -0.04 + 1 * \max \begin{bmatrix} 0.8U^{(0)}(2,3) + 0.1U^{(0)}(1,3) + 0.1U^{(0)}(3,3) & UP \\ 0.8U^{(0)}(1,3) + 0.1U^{(0)}(2,3) + 0.1U^{(0)}(2,3) & LEFT \\ 0.8U^{(0)}(2,3) + 0.1U^{(0)}(1,3) + 0.1U^{(0)}(3,3) & DOWN \\ 0.8U^{(0)}(3,3) + 0.1U^{(0)}(2,3) + 0.1U^{(0)}(2,3) & RIGHT \end{bmatrix}$$

$$U^{(1)}(2,3) = -0.04 + (0.8 * 0.76 + 0.2 * -0.04) = 0.56$$

- Steps two moves away from positive rewards start increasing their value

# State Utilities as Function of Iteration #




یک اشکال عمده :
کار را تاجایی ادامه می دهیم که مقدارها دقیق شده باشند
ولی شاید لازم نباشد برای به دست آوردنpiolicy بهینه تمام مقادیر را بهینه کنیم

- Note that utilities of states at different distances from (4,3) accumulate negative rewards until a path to (4,3) is found

# Policy Iteration

- We saw that we can obtain an optimal policy even when the U function returned by value iteration is not optimal yet
- Intuitively, if one action is clearly better than all others, the precise utilities of the states involved are not necessary to select it به جای تغییر مقادیر در هر iteration ، خود policy را تغییر میدهیم
- This is the intuition behind the ***policy iteration*** algorithm

# Policy Iteration

➢ Algorithm

- $\pi_i \leftarrow$ an arbitrary initial policy, U $\leftarrow$ A vector of utility values, initially 0
- 2. Repeat until no change in $\pi$

  (a) Compute utilities $U^i$ generated by executing $\pi_i$ with current U (***policy evaluation***)

$$U^i(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi_i(s)) U(s')$$

**Watch out**, there is no "MAX" here, because the policy is already fixed!

  (b) Compute a new MEU $\pi_{i+1}$ using a one-step look-ahead based on $U^i$ (***policy improvement***)

Here there is "MAX" because we need to find out whether there is any other action that does better then the action specified by $\pi_i$

For $\forall s$

$$\pi_{i+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s' | s, a) U^{\pi_i}(s')$$

# Policy Iteration

**function** POLICY-ITERATION($mdp$) **returns** a policy
  **inputs:** $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
  **local variables:** $U$, a vector of utilities for states in $S$, initially zero
    $\pi$, a policy vector indexed by state, initially random

چه طور policy را updateکنیم ؟
این policy برای هر state مشخص می کند که باید چه کار کرد
فرض کنیم مشخص نکرده یعنی در هر state کاری که می خواهیم انجام دهیم(اولین کنش) دست خودمان باشد
و بعد ازآن از policyپیروی می کنیم
بنابراین با این روش policy را بهبود می بخشیم
ثابت می شود اگر این کار را تا بی نهایت ادمه دهیم جایی میرسیم که policy ثابت می ماند
و این policy ثابت همان policy بهینه می باشد

بنابراین همواره policy iteration جواب بهینه را می دهد🧐🧐

# Policy Iteration

**function** POLICY-ITERATION($mdp$) **returns** a policy
  **inputs**: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
  **local variables**: $U$, a vector of utilities for states in $S$, initially zero
    $\pi$, a policy vector indexed by state, initially random

  **repeat**
    $U \leftarrow$ POLICY-EVALUATION($\pi$, $U$, $mdp$)
    $unchanged? \leftarrow$ true

# Policy Iteration

**function** POLICY-ITERATION($mdp$) **returns** a policy
  **inputs**: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
  **local variables**: $U$, a vector of utilities for states in $S$, initially zero
    $\pi$, a policy vector indexed by state, initially random

  **repeat**
    $U \leftarrow$ POLICY-EVALUATION($\pi$, $U$, $mdp$)
    $unchanged? \leftarrow$ true
    **for each** state $s$ **in** $S$ **do**
      **if** $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$

# Policy Iteration

**function** POLICY-ITERATION($mdp$) **returns** a policy
  **inputs**: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
  **local variables**: $U$, a vector of utilities for states in $S$, initially zero
    $\pi$, a policy vector indexed by state, initially random

  **repeat**
    $U \leftarrow$ POLICY-EVALUATION($\pi, U, mdp$)
    $unchanged? \leftarrow$ true
    **for each** state $s$ **in** $S$ **do**
      **if** $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  **until** $unchanged?$
  **return** $\pi$