

# Reinforcement Learning

Introduction

# Reinforcement Learning

- Agent interacts and learns from a stochastic environment
- Science of sequential decision making
- Many faces of reinforcement learning
  - Optimal control (Engineering)
  - Dynamic Programming (Operations Research)
  - Reward systems (Neuro-science)
  - Classical/Operant Conditioning (Psychology)

## Characteristics of Reinforcement Learning

- No supervisor, only reward *signals*
- Feedback is delayed
- Sequential decisions
- Actions effect observations (non i.i.d.)

## Examples

- Automated vehicle control
  - An unmanned helicopter learning to fly and perform stunts
- Game playing
  - Playing backgammon, Atari breakout, Tetris, Tic Tac Toe
- Medical treatment planning
  - Planning a sequence of treatments based on the effect of past treatments
- Chat bots
  - Agent figuring out how to make a conversation

# Markov Decision Process (MDP)

# Markov Decision Processes (MDP)

- Sequential decisions in round rounds  $t = 1, \dots, T$
- Important concepts
  - State
  - Action
  - Reward
- Markov property: Future is independent of the past given the current state

# Markov Decision Processes (MDP)

- Starts at some initial state  $s_1$
- In every round  $t$ , the agent
  - observes the current state  $s_t$ ,
  - take an action  $a_t$ , and then
  - observes a reward signal  $r_t$
  - transitions to the next state  $s_{t+1}$
- Markov Property:
  - $\Pr(s_{t+1} = s' | \text{history till time } t) = \Pr(s_t = s' | s_t = s, a_t = a) =: P_{s,a}(s')$
  - $E[r_t | \text{history till time } t] = E[r_t | s_t = s, a_t = a] =: R_{s,a}$

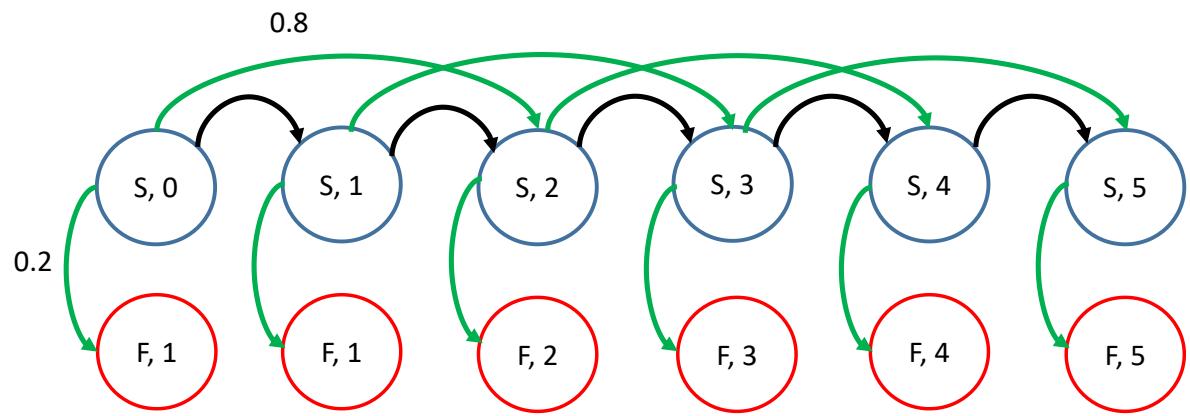
# Markov Decision Processes (MDP)

- Goal: Maximize some form of cumulative reward
- Total reward in finite time T
  - maximize  $\sum_{t=1}^T r_t$
- Infinite time average reward
  - maximize  $\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T r_t$
- **Discounted sum of rewards**
  - maximize  $r_1 + \gamma r_2 + \gamma^2 r_3 + \dots + \gamma^{i-1} r_i + \dots$
  - where  $\gamma < 1$

## Summary

- Markov Decision Process (MDP) is a tuple  $(S, s_1, A, P, R)$
- $S$  is a finite set of states
- $A$  is a finite set of actions
- $P$  is a state transition probability matrix of dimension  $S \times A \times S$ 
$$P_{s,a}(s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$
- $R$  is a reward function
$$R_{s,a} = Ex[r_t | s_t = s, a_t = a]$$
- Goal definition, discount factor  $\gamma \in [0,1)$

## Example



# Markov Decision Processes

Finding an optimal policy: Value functions

## Overview

- Markov Decision Process is a tuple  $(S, s_1, A, P, R)$
- $P$  is a state transition probability matrix of dimension  $S \times A \times S$

$$P_{s,a}(s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

- $R$  is a reward function

$$R_{s,a} = E[R_{t+1} | s_t = s, a_t = a]$$

- Goal:

Maximize expected discounted reward  $E[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_1]$

where  $r_t = R_{s_t, a_t}$ ,  $\gamma \in [0, 1)$  is a discount factor

## Policy

- A policy  $\pi: S \rightarrow A$  is a mapping from state space to action space
- Following a stationary policy  $\pi$  means taking action  $a_t = \pi(s_t)$  at all time steps  $t$
- Theorem

For any discounted MDP, there always exists stationary policy  $\pi$  that is optimal

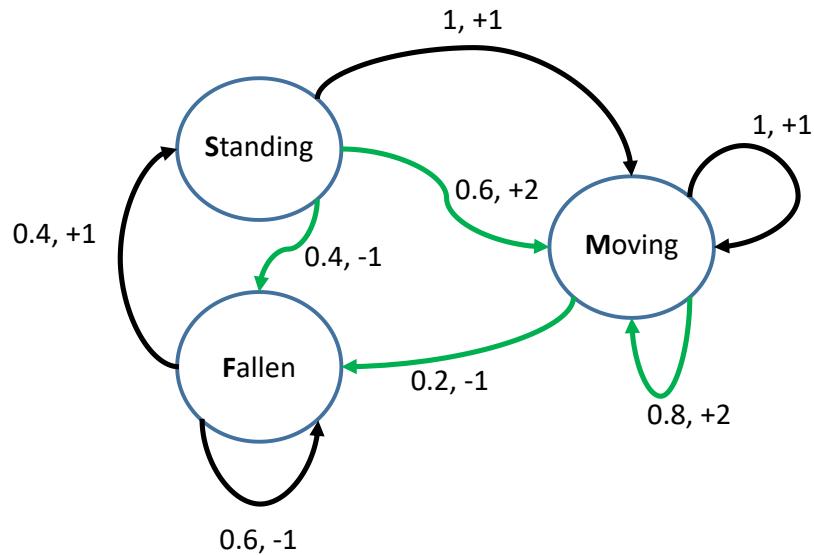
## Value function

- Value function  $v_\pi(s)$  of a policy  $\pi$ 
  - expected reward starting from state  $s$  and then following the policy  $\pi$

$$v_\pi(s) = E[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_1 = s]$$

where  $a_t = \pi(s_t)$ ,  $E[r_t | s_t, a_t] = R_{s_t, a_t}$ ,  $\Pr(\cdot | s_t, a_t) = P_{s_t, a_t}$

## Example



Policy: slow action 1 (black) in *Fallen* state, fast action 2 (green) in *Standing* and *Moving* state

## Bellman equations

- Value function can be decomposed into immediate reward plus discounted value function of the next state

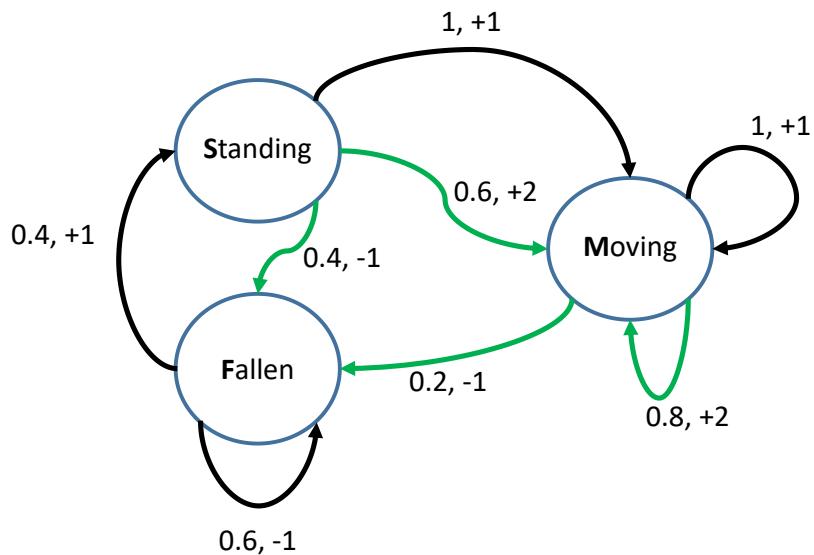
$$v_{\pi}(s) = R_{s,\pi(s)} + \gamma \sum_{s'} P_{s,\pi(s)}(s') v_{\pi}(s')$$

- Compact matrix notation

$$\mathbf{v}_{\pi} = \mathbf{r}_{\pi} + \gamma P_{\pi} \mathbf{v}_{\pi}$$

$$\mathbf{v}_{\pi} = (\mathbf{I} - \gamma P_{\pi})^{-1} \mathbf{r}_{\pi}$$

## Example



Policy: slow action 1 (black) in *Fallen* state, fast action 2 (green) in *Standing* and *Moving* state

# Markov Decision Processes

Finding an optimal policy: Iterative methods

## Recap

- Value function  $v_\pi(s)$  of a policy  $\pi$

$$v_\pi(s) = E[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots | s_1 = s]$$

- Bellman equations

$$\mathbf{v}_\pi = \mathbf{r}_\pi + \gamma P_\pi \mathbf{v}_\pi$$

$$\mathbf{v}_\pi = (\mathbf{I} - \gamma P_\pi)^{-1} \mathbf{r}_\pi$$

## Optimal Policy

- Optimal policy when starting in state  $s$ :

$$\operatorname{argmax}_{\pi} v_{\pi}(s)$$

# Optimal Policy

- Define partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s$$

- Theorem

- There always exists a policy that is better than all other policies

$$\pi \geq \pi' \text{ for all } \pi'$$

Such a policy is called an optimal policy

- All optimal policies achieve the same value function  $v_*(s)$  called the optimal value function

## Bellman Optimality Equations

- Optimal value functions are recursively related by Bellman optimality equations

$$v_*(s) = \max_{a \in A} R_{s,a} + \gamma \sum_{s'} P_{s,a}(s') v_*(s')$$

- Matrix notation

$$\mathbf{v}_* = \max_{\pi} \mathbf{r}_{\pi} + \gamma \mathbf{P}_{\pi} \mathbf{v}_*$$

- Optimal policy can be computed by solving Bellman equations

## Solving the Bellman optimality equations

- No closed form solution in general
- Iterative solution methods
  - Policy iteration
  - Value Iteration

## Policy Iteration

- Start with a random policy  $\pi$

In every iteration,

- Evaluate the policy

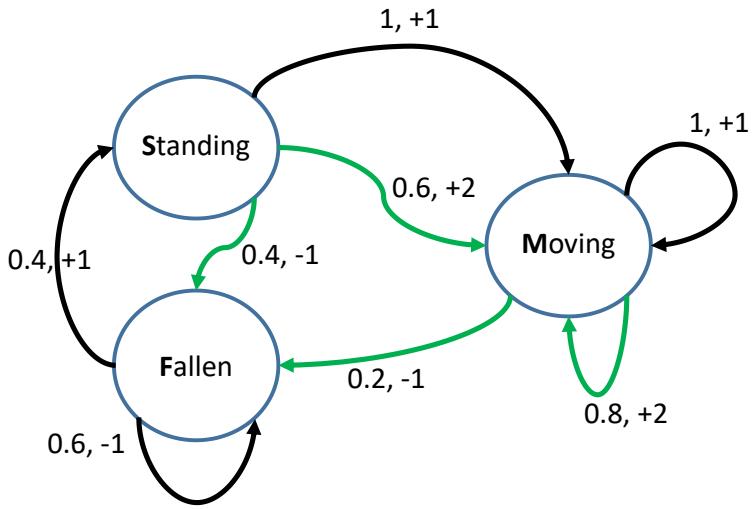
- Compute the value vector for  $\mathbf{v}_\pi = (\mathbf{I} - \mathbf{P}_\pi)^{-1} \mathbf{r}_\pi$

- Improve the policy

- New policy:  $\pi'(s) = \arg \max_a R_{s,a} + \gamma P_{s,a} \mathbf{v}_\pi$

- Stop if no strict improvement ( $\mathbf{v}_\pi = \mathbf{v}_{\pi'}$ )

$$v_\pi(s) = \max_a R_{s,a} + \gamma P_{s,a} \mathbf{v}_\pi , \forall s$$



Starting Policy: always slow action

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \gamma = 0.1$$

- Iteration 1

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix}$$

Improve policy:

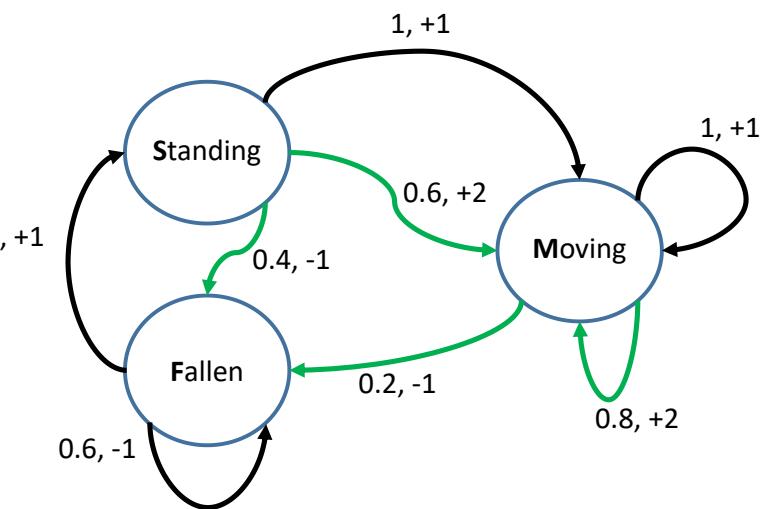
Compute  $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

**State Standing,**

**Slow Action:**  $= 1.11$

**Fast Action:**  $0.8 +$

$$0.1 [0.4 \quad 0 \quad 0.6] \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix} = 0.86$$



Starting Policy: always slow action

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- Iteration 1

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix}$$

Improve policy:

$$\text{Compute } \arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$$

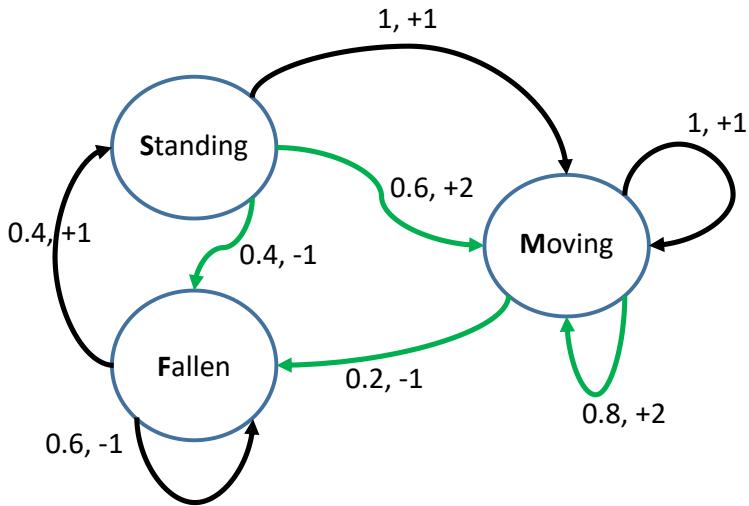
**State Standing, slow action**

**State Moving**

**Slow Action:**  $= 1.1111$

**Fast Action:**  $1.4 +$

$$0.1 [0.2 \ 0 \ 0.8] \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix} \approx 1.48$$



Starting Policy: always slow action

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- Iteration 1

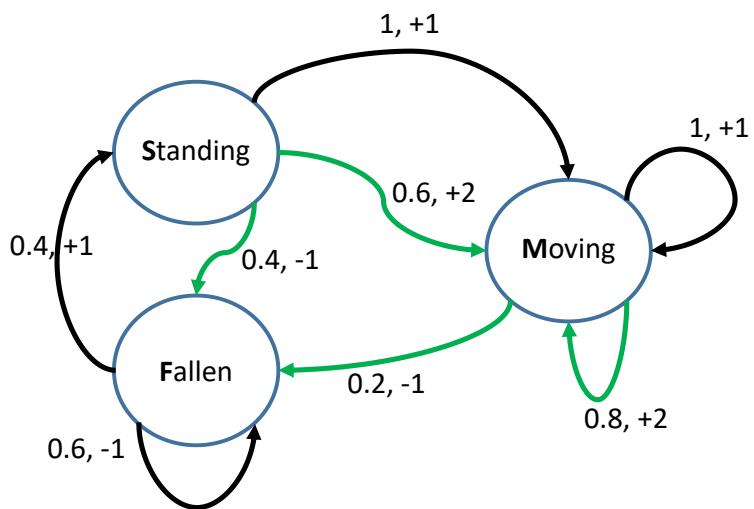
$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1655 \\ 1.1111 \\ 1.1111 \end{bmatrix}$$

Improve policy:

Compute  $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

State *Standing*, SLOW action

State *Moving*, FAST action



**New Policy:** fast action in moving state, slow elsewhere

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1.4 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0.2 & 0 & 0.8 \end{bmatrix}$$

- Iteration 2

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix}$$

Improve policy:

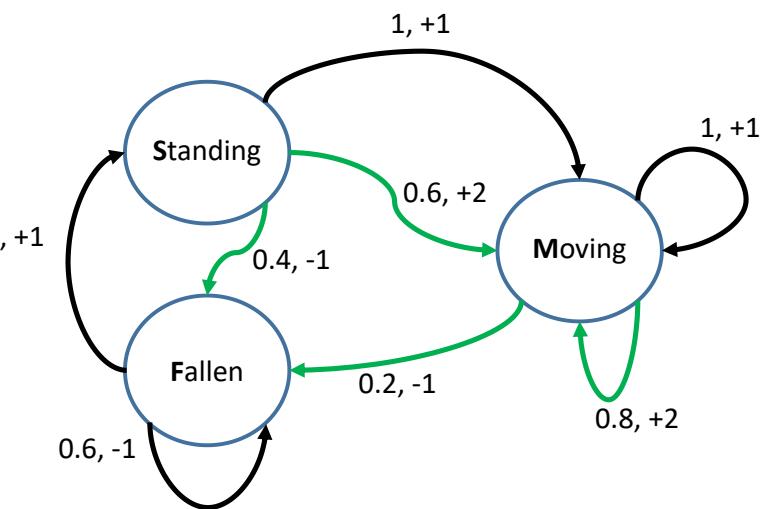
$$\text{Compute } \arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$$

**State Standing,**

**Slow Action:**  $= 1.1518$

**Fast Action:**  $0.8 +$

$$0.1 [0.4 \quad 0 \quad 0.6] \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix} \simeq 0.88$$



**New Policy:** fast action in moving state, slow elsewhere

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1.4 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0.2 & 0 & 0.8 \end{bmatrix}$$

- Iteration 2 Improve policy:

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix}$$

Compute  $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

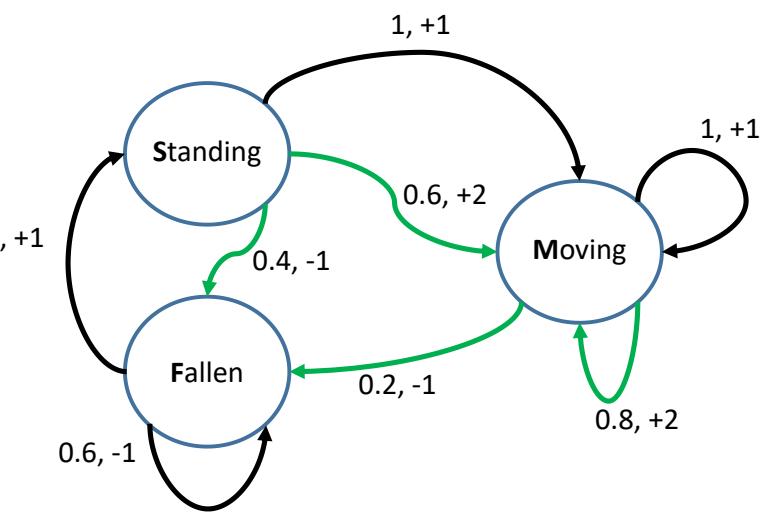
State *Standing*: SLOW action

State *Moving*,

Fast Action: =1.5182

Slow Action: 1 +

$$0.1 [0 \ 0 \ 1] \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix} \simeq 1.1518$$



**New Policy:** fast action in moving state, slow elsewhere

$$r_\pi = \begin{bmatrix} -0.2 \\ 1 \\ 1.4 \end{bmatrix} \quad P_\pi = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0.2 & 0 & 0.8 \end{bmatrix}$$

- Iteration 2 Improve policy:

$$v_\pi = (I - P_\pi)^{-1} r_\pi = \begin{bmatrix} -0.1638 \\ 1.1518 \\ 1.5182 \end{bmatrix}$$

Compute  $\arg \max_a R_{s,a} + \gamma P_{s,a} v_\pi$

State *Standing*: SLOW action

State *Moving*, FAST action

New policy is the same as the old policy

**STOP!**

## Value Iteration method

- Finding optimal value function
  - No explicit policy
- In every iteration  $k$ , improve the value vector

$$v^{(k+1)}(s) = \max_a R_{s,a} + \gamma P_{s,a} v^{(k)}$$

- Converges to  $v_*$

$$v^{(k)} \rightarrow v_*$$

- Optimal policy given by  $\max_a R_{s,a} + \gamma P_{s,a} v_*$

# Reinforcement Learning

## Algorithms

## Model free methods

- Reinforcement learning  $\equiv$  MDP with unknown transition model and/or reward distribution
- Model is unknown but agent observes samples
- Learn while optimizing the policy

## Formulation

- Starts at some initial state  $s_1$

In every round  $t$ , the agent

- observes the current state  $s_t$ ,
- take an action  $a_t$ , and then
- observes a reward signal  $r_t$ , and next state  $s_{t+1}$

$$E[r_t | s_t = s, a_t = a] = R_{s,a}$$

$$\Pr(s_{t+1} = s' | s_t = s, a_t = a) = P_{s,a}(s')$$

$\{R_{s,a}, P_{s,a}\}$  are unknown

# Goal

- Find the optimal policy:

Policy that maximizes expected sum of discounted reward

$\{R_{s,a}, P_{s,a}\}$  are unknown

## Q-learning

- Uses “Q-values” instead of value function
- $Q(s, a)$ : the value of taking action  $a$  in state  $s$
- Formally

$$Q(s, a) = R_{s,a} + \gamma E_{s'}[\max_{a'} Q(s', a')]$$

Immediate expected reward plus the best utility from the next state onwards.

- From Bellman optimality equations, an optimal policy  $\pi$  satisfies

$$Q(s, \pi(s)) = R_{s,\pi(s)} + \gamma E_{s'}[Q(s', \pi(s'))] = v_*(s)$$

## Q-learning

- Proceeds in discrete rounds  $t = 1, 2, \dots$

**In every round  $t$ ,**

- Choose action greedily using “estimated” Q-values

$$a_t = \operatorname{argmax}_a \hat{Q}(s_t, a)$$

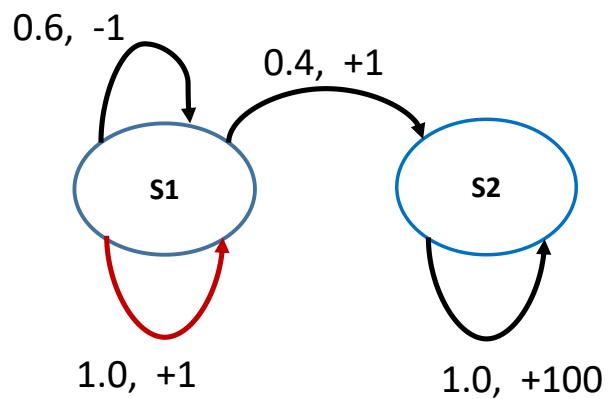
- Take action  $a_t$  observe reward  $r_t$ , next state  $s_{t+1}$

- Update Q-values for  $s_t, a_t$

$$\hat{Q}(s_t, a_t) = r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

(Compare to  $Q(s, a) = R_{s,a} + \gamma E_{s'}[\max_{a'} Q(s', a')]$ )

## The need for Exploration



## Epsilon Greedy exploration

- With probability  $1 - \epsilon$ , use greedy action

$$a_t = \operatorname{argmax}_a \hat{Q}(s_t, a)$$

- With probability  $\epsilon$ , play random action