Reinforcement Learning

IE562 Computational Foundations of Smart Systems

Juxihong Julaiti

Sep, 2018

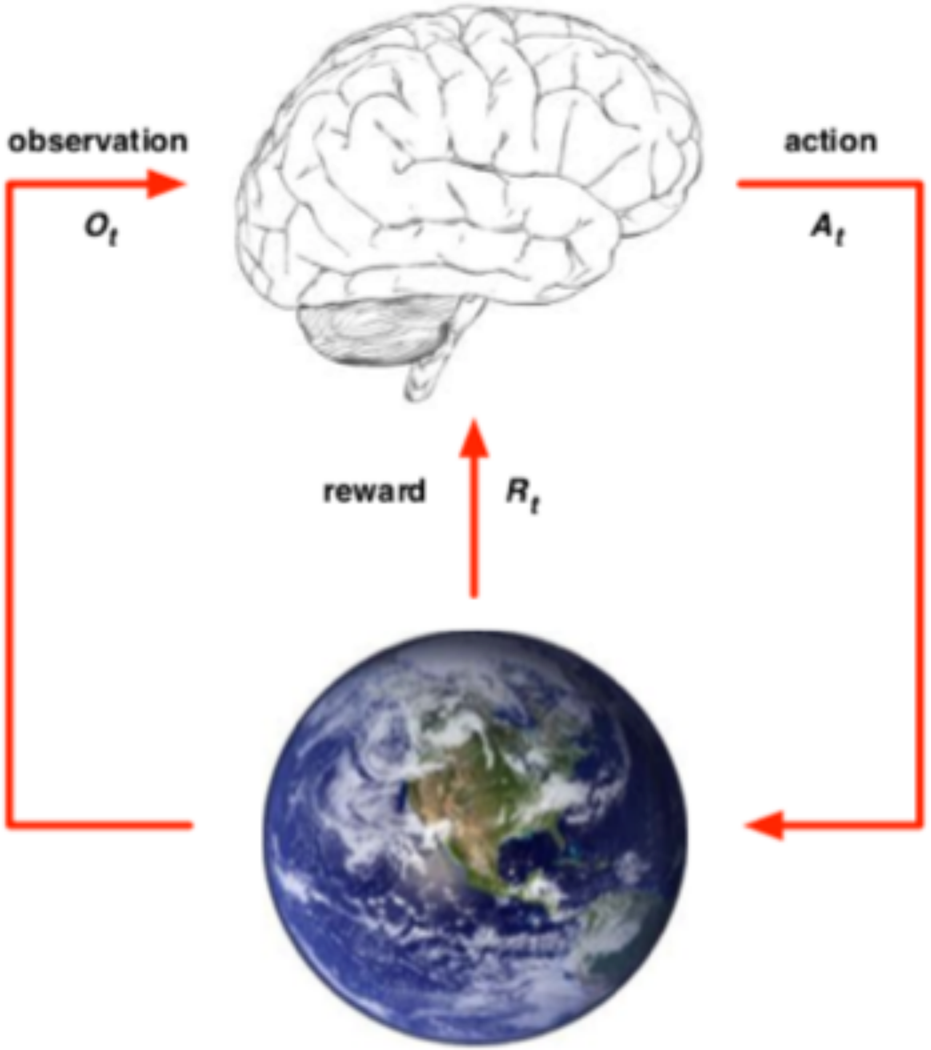
Part A

Outline

- Reinforcement Learning (RL)
- What is RL
- Markov Processes and Markov Decision Processes
- Value Functions
- Bellman Expectation Equation
- Optimal Value Function
- Solving the Bellman Optimality Equation
- Evaluating a Random Policy in the Small Gridworld
- Extensions to MDPs

Reinforcement Learning (RL)

What is RL: Agent and Environment



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Goal: find a policy that enables the agent to get the maximum cumulative rewards

What is RL: Policy and Cumulative Reward

- **A policy $\boldsymbol{\pi}$,** is a mapping from a state to actions:

$$\pi(a|s) = P[A_t = a|S_t = s]$$

For instance, $s \in [0,1,2,\ldots,N]$, number of jobs in the queue, and a is the speed of the machine, $a \in Z^+$ a policy can be:

$$\pi(a|0) = \begin{cases} 0, a > 0 \\ 1, a = 0 \end{cases}$$

$$\pi(a|1) = \begin{cases} 0, a = 0 \\ e^{-a}, a > 0 \end{cases}$$

Cumulative reward: $\mathrm{G}_0 = \mathrm{R}_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T$

Goal:

$$\max_{\pi} \sum_{t} \gamma^{t-1} R_t$$

How to solve it?
Markov Processes and Markov Decision Processes

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, ...$ with the Markov property.

Definition

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$

How to solve it?
Markov Processes and Markov Decision Processes

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$.

Value Functions

Definition

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

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

Definition

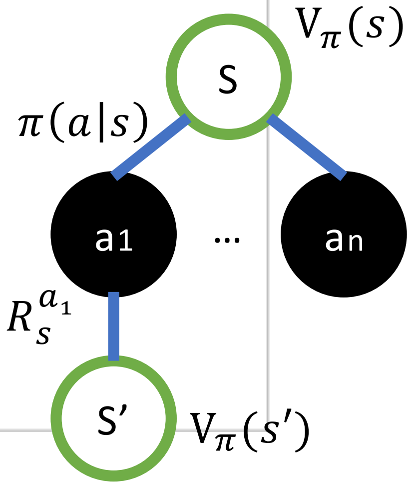
The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

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

Bellman Expectation Equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

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



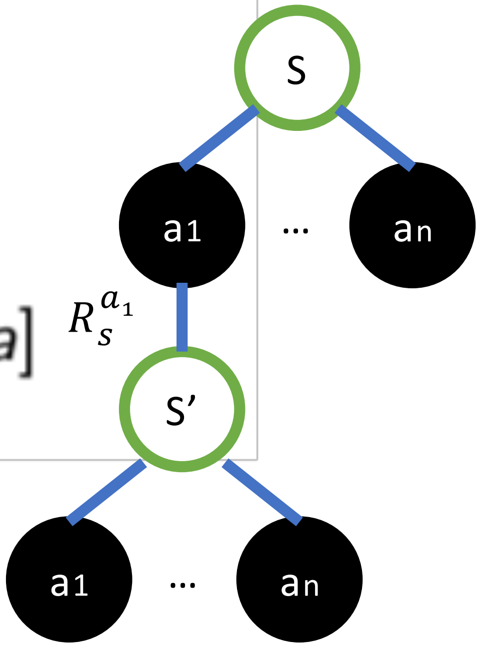
If we have all the info. about the MDP:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s)\left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s')\right]$$

Bellman Expectation Equation (Cont.)

The action-value function can similarly be decomposed,

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



If we have all the info. about the MDP:

$$q_\pi(s,a) = R_s^a + \sum_{s'} P_{ss'}^a \sum_{a'} \pi(a'|s') q_\pi(s', a')$$

Optimal Value Function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

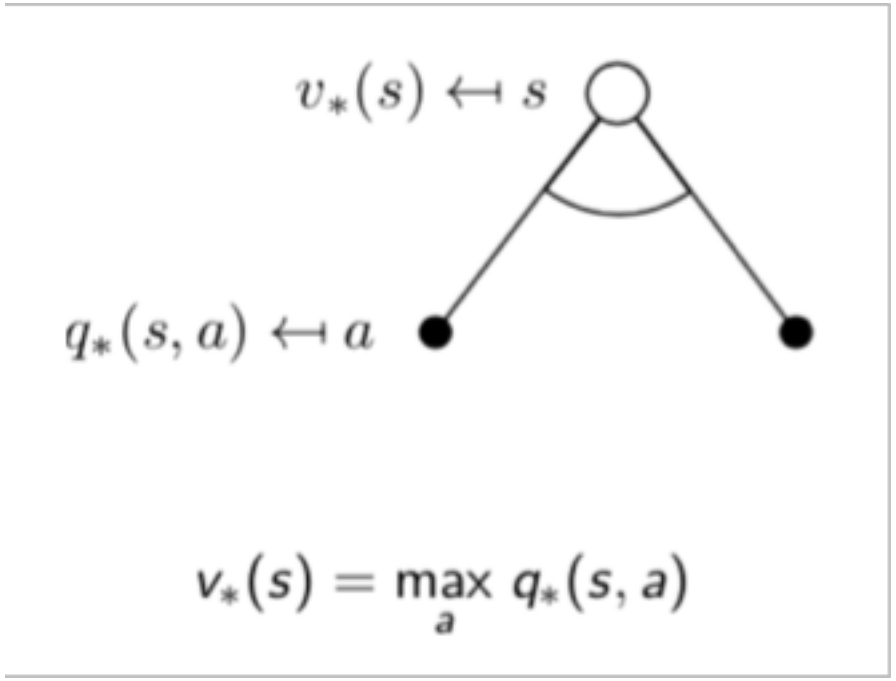
$$v_*(s) = \max_{\pi} v_\pi(s)$$

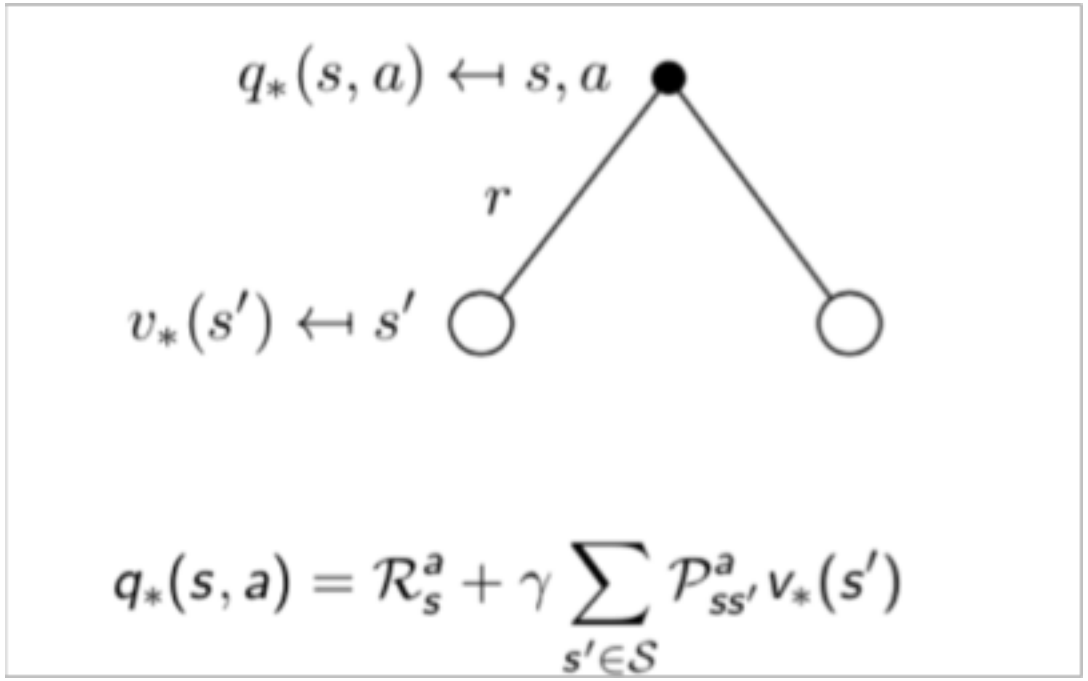
The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_{\pi} q_\pi(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

Optimal Value Function (Cont.)





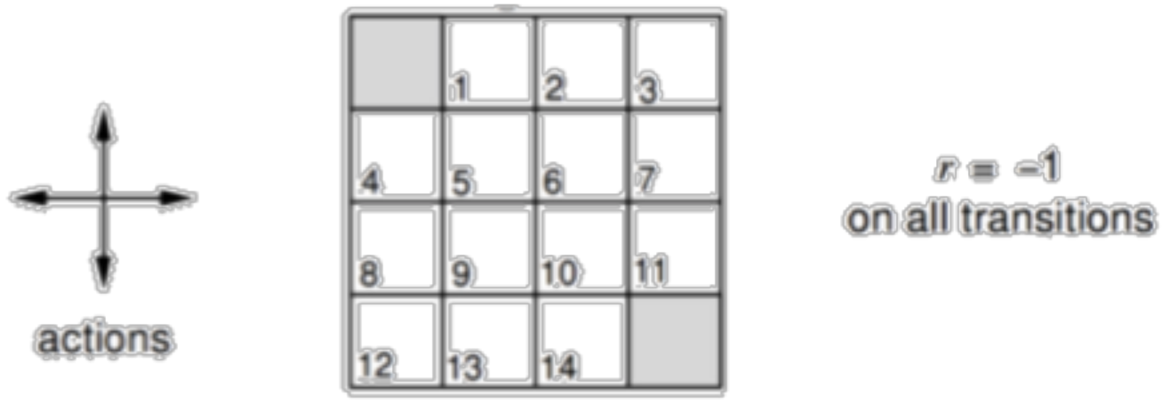
$$V_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s')$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max q_*(s', a')$$

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

Evaluating a Random Policy in the Small Gridworld



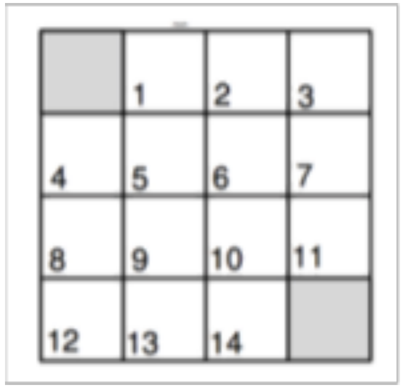
- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \ldots, 14$
- Two terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

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

Evaluating a Random Policy in the Small Gridworld

Goal: given a policy find V(s) for all s

v_k for the Random Policy



Initiate v(s) as 0 for all s

$k=0$

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

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s') \right]$$

$V_\pi(1)$=0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)=-1

$V_\pi(2)$=0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)=-1

$V_\pi(3)$=0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)+ 0.25*(-1+1*1*0)=-1

…

$k=1$

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

$V_\pi(1)$=0.25*(-1+1*1*0)+ 0.25*(-1+1*1*-1)+ 0.25*(-1+1*1*-1)+ 0.25*(-1+1*1*-1)

=-0.25+3*0.25*-2=-1.75

…

$V_\pi(5)$=0.25*(-1+1*1*-1)+ 0.25*(-1+1*1*-1)+ 0.25*(-1+1*1*-1)+ 0.25*(-1+1*1*-1)

=4*0.25*-2=-2

…

$k=2$

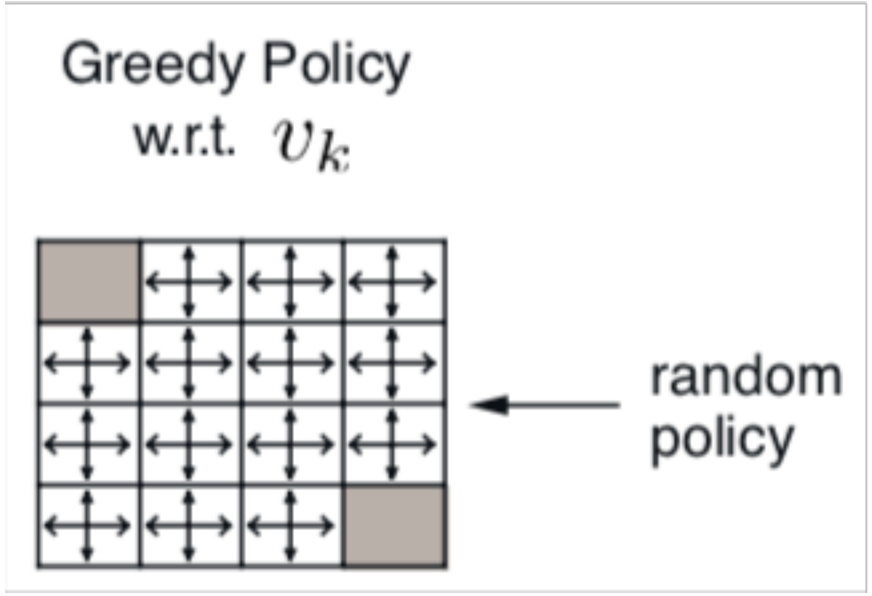
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

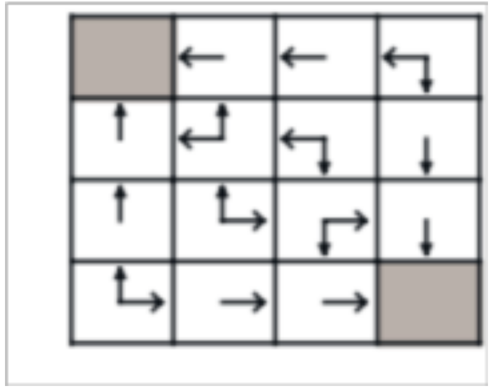
Evaluating a Random Policy in the Small Gridworld

Goal: given a policy find V(s) for all s

$k = \infty$

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





Optimal Policy

Extensions to MDPs

- Infinite and continuous MDPs
- Partially observable MDPs
- Undiscounted, average reward MDPs

Reference:

- Reinforcement Learning: An Introduction

  By Richard S. Sutton and Andrew G. Barto

  http://incompleteideas.net/book/bookdraft2017nov5.pdf

Part B

Outline

- Recap
- Categorizing RL Methods
- Model-based:
  - Policy-based Methods for Planning
  - Value-based Methods for Planning
  - Challenges
- Model-free:
  - Policy-based Methods for Learning
    - Monte-Carlo Methods
    - Temporal-Difference Methods
  - Value-based Methods for Learning
    - SARSA
    - Q-learning
- More

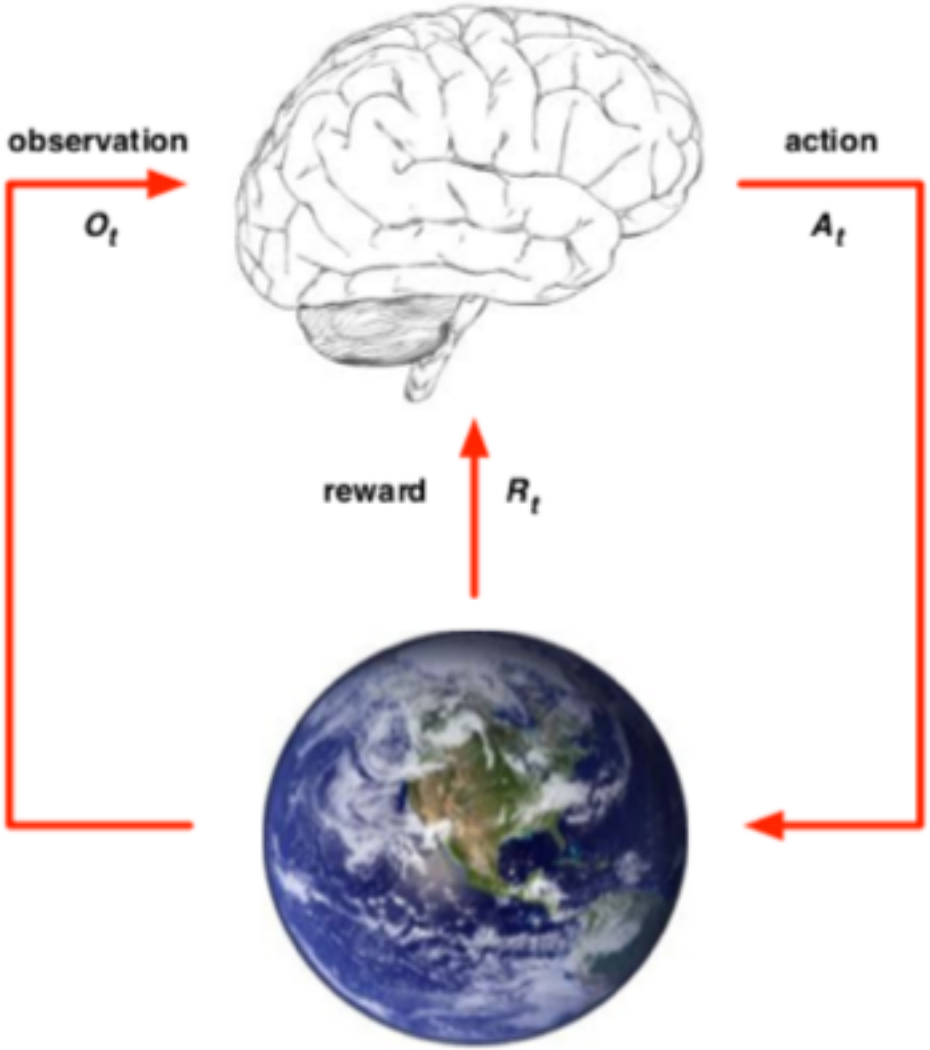
Recap
Supervised, Unsupervised Learning and RL

- You have a **target, a value or a class to predict**. A model is trained to minimize a loss function to minimize the cross entropy or mean squared error between predictions and true values

- You have **unlabelled data**, **unsupervised machine learning** will group the data by minimizing a given loss function (minimize the sum of distances between the central of each cluster and elements inside the cluster)

- You want **to attain an objective (maximize a reward signal or minimize penalty), reinforcement learning** will play this game many times to find the best policy

Recap: RL

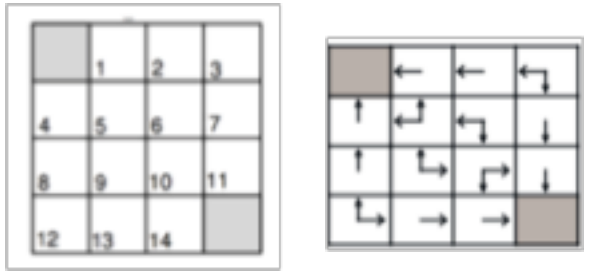


- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Goal: find a policy that enables the agent to get the maximum cumulative rewards

Categorizing RL Methods

1. Policy-based methods (PBMs)
2. Value-based methods (VBMs)
3. Actor Critic



Two types of problems:

The Planning Problem:

- A model of the environment is known (**Model-based**)

PBM

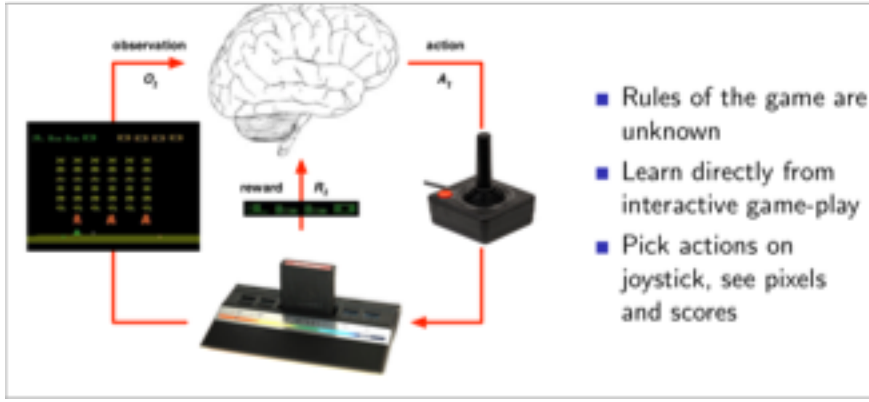
1. Compute values function given a policy
2. Improve the policy , go back to 1 until the policy is converged

The Learning Problem:

- The environment is initially unknown (**Model-free**)

PBM

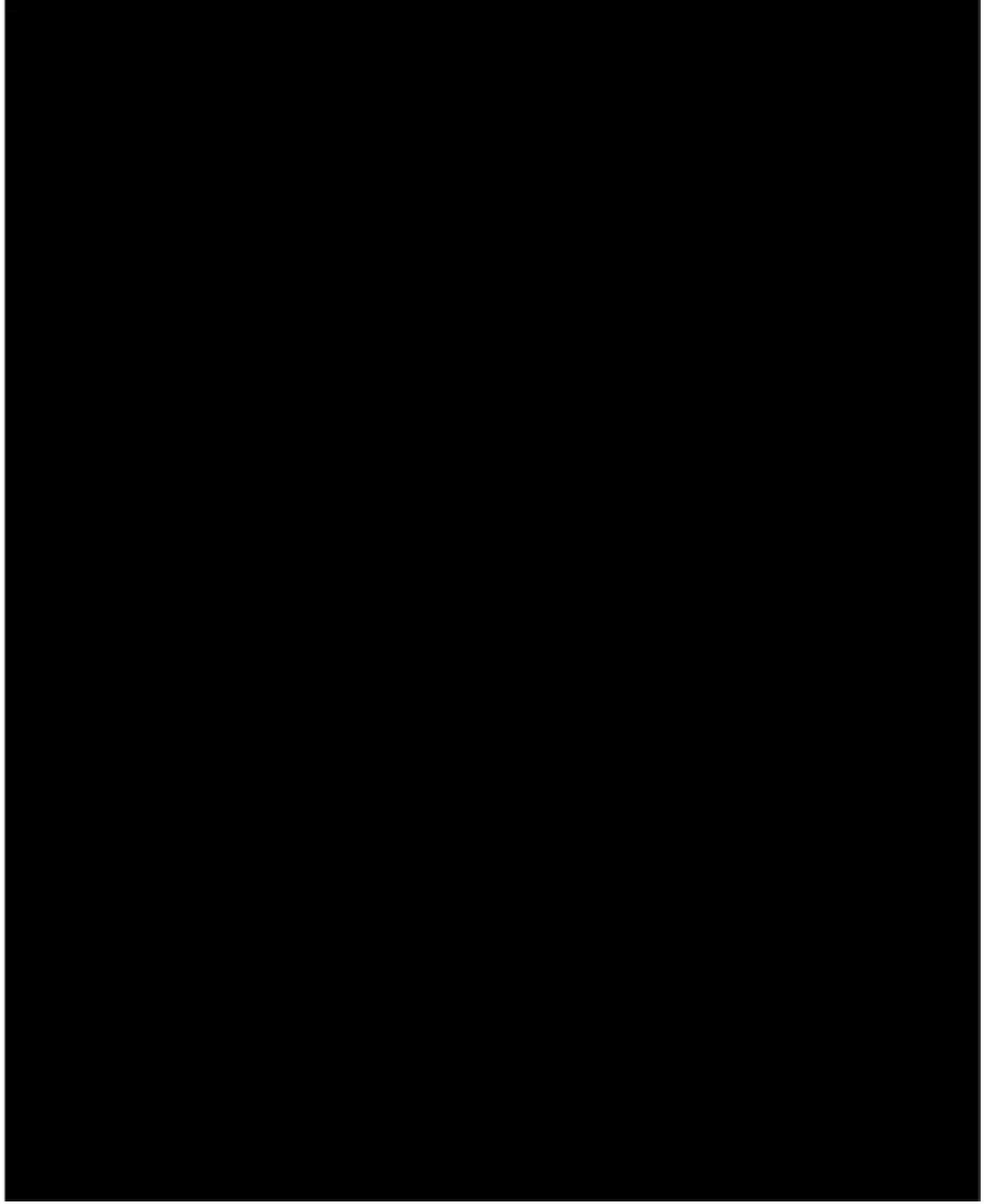
1. Interacts with the environment
2. Improve the policy, go back to 1 until the policy is converged



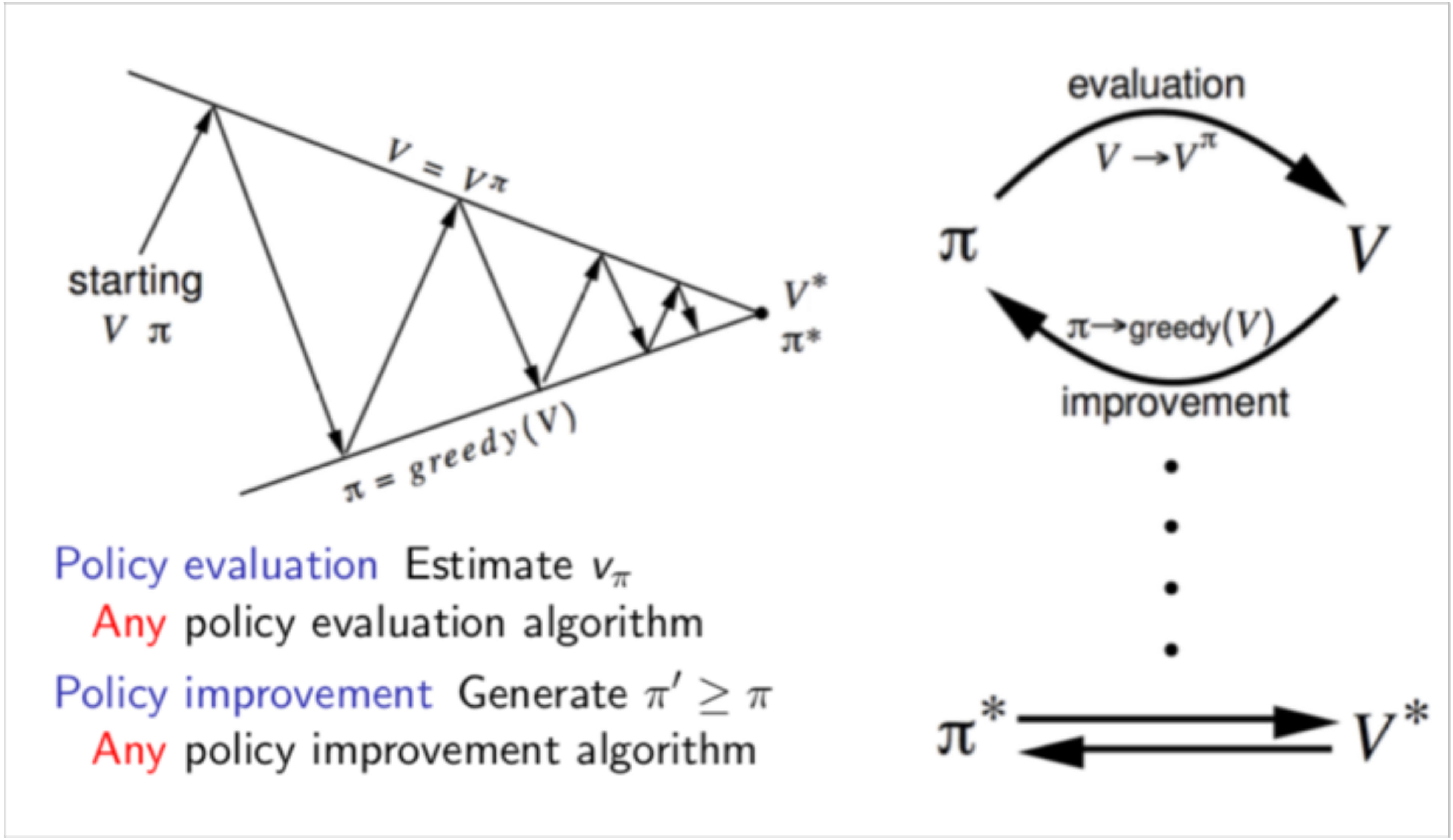
When the MDP is complex, a planning problem can be treated as a learning problem

Atari breakout: An Example of Learning (model-free)

Human-level control through deep reinforcement learning



Policy-based Methods for Planning



Value-based Methods for Planning

- Synchronous value iteration stores two copies of value function

  for all s in $\mathcal{S}$

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

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

- In-place value iteration only stores one copy of value function

  for all s in $\mathcal{S}$

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

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

$k = 1$

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

Both methods will converge to v* (check the book)

Value-based Methods for Planning

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

Model-based: Challenges

- For large problems DP suffers Bellman’s curse of dimensionality
  - Numbers of states n grows exponentially with number of state variables

- What we do? Treat it as a learning problem!

Policy-based Methods for Learning

- Policy-based Methods
  - Policy evaluation: evaluating value functions given a policy
  - Policy improvement: improve the policy by acting greedily in terms of the values
- Hard part is the evaluation
- Methods:
  - Monte-Carlo methods
  - Temporal-Difference methods

Monte-Carlo Methods

- Goal: learn v_π from episodes of experience under policy $\boxed{\pi}$

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

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

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

Monte-Carlo Methods

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to *episodic* MDPs
    - All episodes must terminate

Monte-Carlo Methods (Cont.)

- **First-Visit** Monte-Carlo Policy Evaluation

  - To evaluate state s
  - The first time-step t that state s is visited in an episode,
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$
  - By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Monte-Carlo Methods (Cont.)

- **Every-Visit** Monte-Carlo Policy Evaluation

- To evaluate state s
- Every time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Temporal-Difference Methods

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

Temporal-Difference Methods (Cont.)

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Advantages and Disadvantages of MC and TD

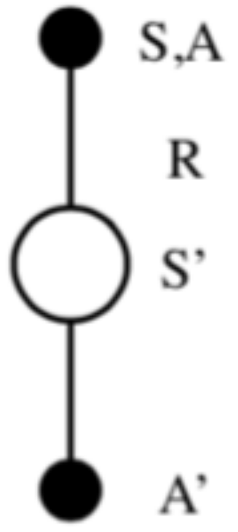
- TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Value-based Methods for Learning

- There is no a fixed policy
  - Instead, the agent will always act greedily in terms of Q(s,a)

- Methods:
  - SARSA
  - Q-learning

SARSA

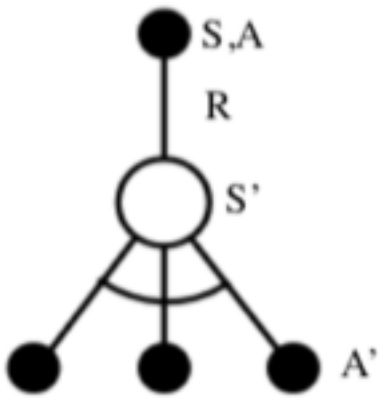


$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma Q(S',A') - Q(S,A)\right)$$

SARSA (Cont.)

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma Q(S', A') - Q(S, A)\right]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Q-Learning



$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S',a') - Q(S,A) \right)$$

Theorem

Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow q_*(s,a)$

Q-Learning (Cont.)

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
        $S \leftarrow S'$;
    until S is terminal

Exploration and Exploitation

- Restaurant Selection
  - Exploitation Go to your favourite restaurant
  - Exploration Try a new restaurant

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit

More

- TD-lambda
- N-step SARSA
- SARSA-lambda
- Deep Q-learning
- Eligibility trace
- Important sampling
- Actor crtic
- ….