Introduction to Machine Learning

Lecture 15: Elementary Reinforcement Learning – Deterministic Environment

Dec 2, 2019

Jie Wang

Machine Intelligence Research and Applications Lab

Department of Electronic Engineering and Information Science (EEIS)

http://staff.ustc.edu.cn/~jwangx/

jiewangx@ustc.edu.cn

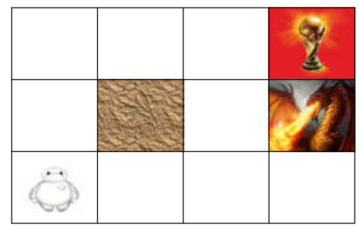
MIRA Machine Intelligence Research and Applications Lab

Contents

- Learning Scenarios
- Markov Decision Process
- Planning Algorithms
- Learning Algorithms

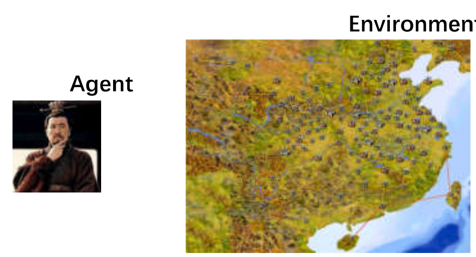
Learning Scenarios

Grid World

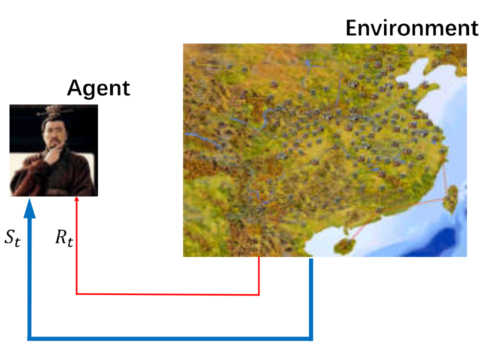


Snooker

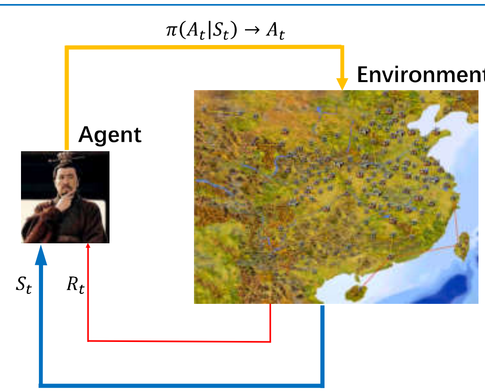
Three Kingdoms



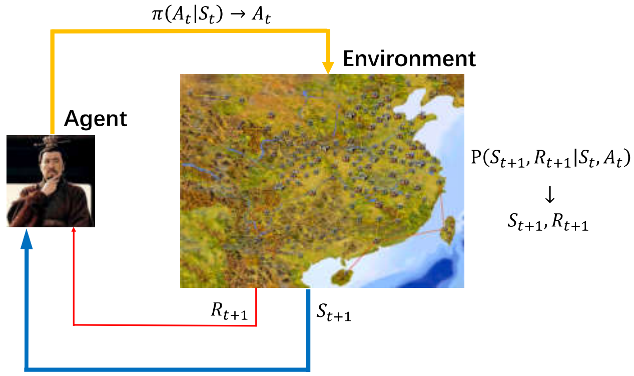
Three Kingdoms



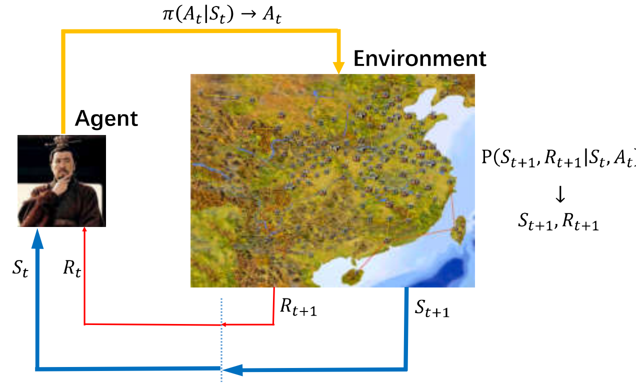
Three Kingdoms



Three Kingdoms



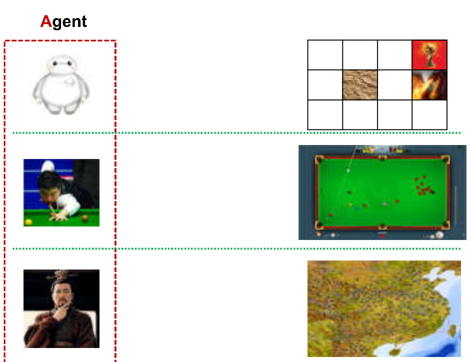
Three Kingdoms



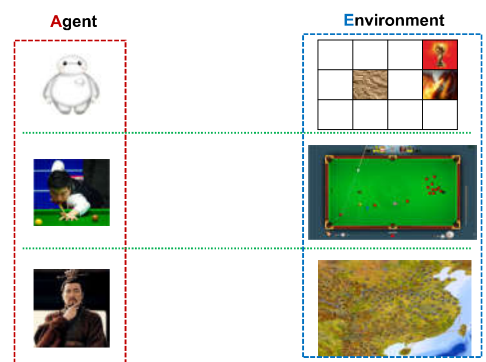
Agent & Environment

Agent & Environment

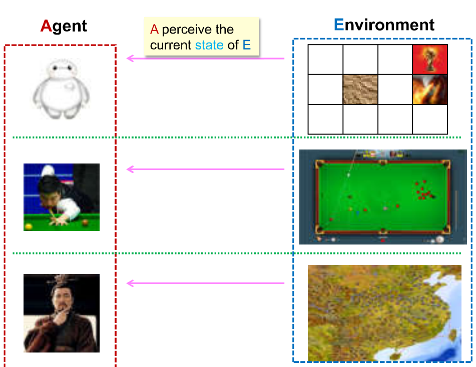
Agent & Environment



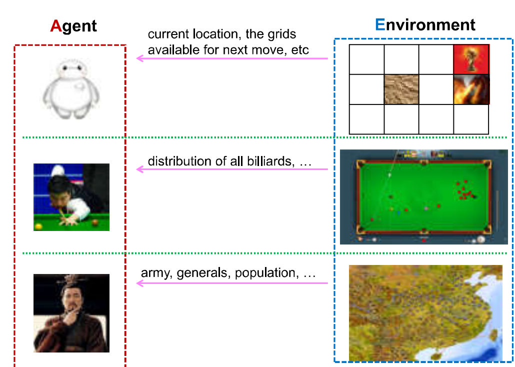
Agent & Environment



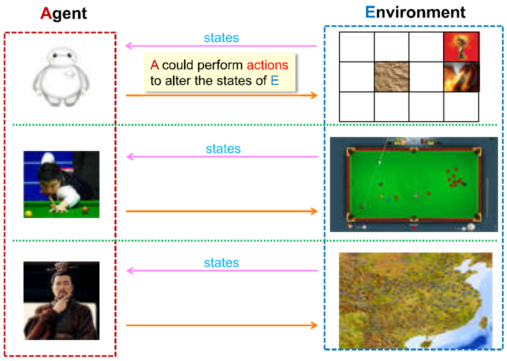
Interactions between Agent & Environment



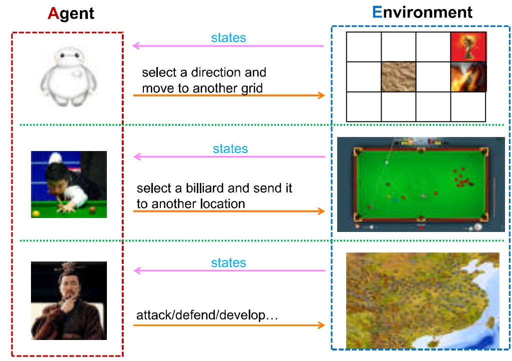
Interactions between Agent & Environment



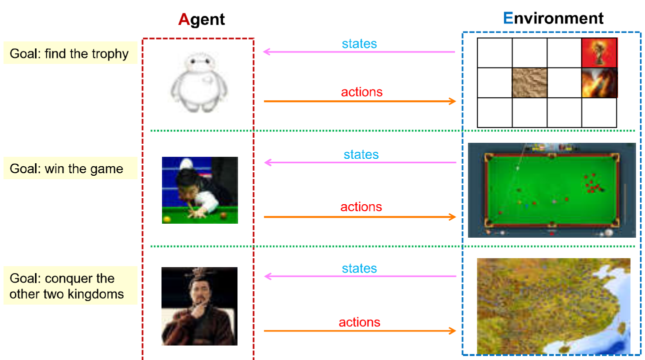
Interactions between Agent & Environment



Interactions between Agent & Environment



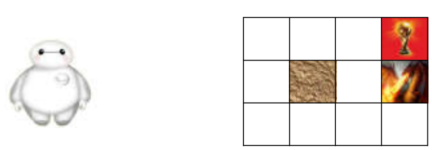
Goal State of the Agent



Markov Decision Process

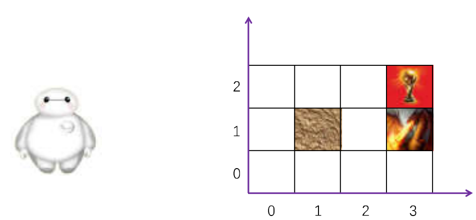
Agent & Environment

- The system consists of an agent (may be more) and an environment, interacting with each other.



States

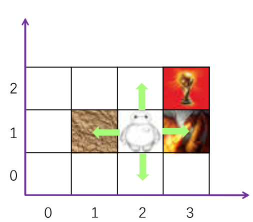
- From the perspective of the agent, the environment is described by a set of states.



States: $\mathcal{S} = \{(i,j) : i = 0,1,2,3, j = 0,1,2\}$

Actions

- At each state, the agent can pick and perform certain action to alter the state.



$\delta((2,1), \mathrm{up}) = (2,2)$

$\delta((2,1), \mathrm{down}) = (2,0)$

$\delta((2,1), \mathrm{left}) = (2,1)$

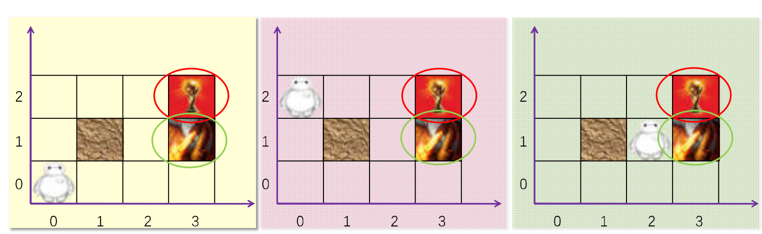
$\delta((2,1), \mathrm{right}) = (3,1)$

$\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$: deterministic environment

Action: $\mathcal{A} = \{\mathrm{up}, \mathrm{down}, \mathrm{left}, \mathrm{right}\}$

Goal State

- No matter starting from which state, the agent would like to achieve certain goal state.

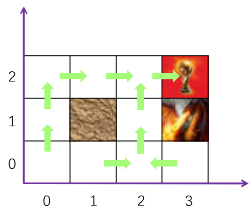


The game will terminate if the agent arrives at $(3,2)$ (win) or $(3,1)$ (lose).

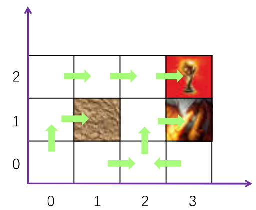
The states $(3,2)$ and $(3,1)$ are also called absorbing states

Policy

- To achieve the goal state, the agent needs to pick and perform a sequence of actions according to the observed states.



A good policy

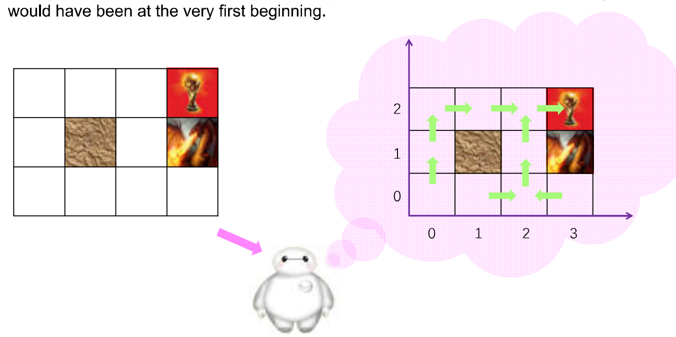


A bad policy

Policy: $\pi : \mathcal{S} \to \mathcal{A}$

The Learning Task

- Find a policy that can direct the agent to its goal state no matter which state the agent would have been at the very first beginning.



The Learning Task

How can we find a desired policy to direct the agent's move?

Reward

- We assume that the goals of the agent can be defined by a reward function

$$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

The reward function is not always available. For some applications, you need to define it properly.

- Starting from an arbitrary state, the desired policy would pick for the agent the actions that maximize the reward accumulated over time.

Looking for a policy that would pick for the agent the actions to achieve its goals

starting from an arbitrary state

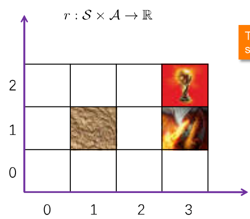
Looking for a policy to pick for the agent the actions that will maximize the accumulated reward over time

Reward

- We assume that the goals of the agent can be defined by a reward function

$$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

The reward function is not always available. For some applications, you need to define it properly.



$$r(s,a) = \begin{cases} 100, & \text{if } \delta(s,a) = (3,2) \\ -100, & \text{if } \delta(s,a) = (3,1) \\ 0, & \text{otherwise} \end{cases}$$

Markov Decision Process (MDP)

- Indeed, we have already introduced the so-called MDP, which is defined (rigorously) by
  - a set of states $\mathcal{S}$, possibly infinite
  - a set of actions $\mathcal{A}$, possibly infinite
  - an initial state $s_0 \in \mathcal{S}$
  - a transition probability $\mathbf{P}[s'|s,a]$: distribution over destination states $s' = \delta(s,a)$
  - a reward probability $\mathbf{P}[r|s,a]$: distribution over rewards $r' = r(s,a)$

MRT Chapter 14

- This model is Markovian because the transition and reward probabilities only depend on the current state and the action picked and performed at the current state, instead of the previous sequence of states and actions performed.
- In this lecture, we assume that
  - the states and the actions are finite
  - the environment is **deterministic**, i.e., the destination state and the reward are completely determined by the current state and the action performed at the current state

The Optimal Policy

Under a MDP, we shall look for the (optimal) policy that leads to the greatest (expected) accumulated reward no matter which state the agent begins with.

Value Function

- Suppose that a policy π is given.
- Starting from an arbitrary state s_t, the cumulative reward by following π is given by

$$V^\pi(s_t) := r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

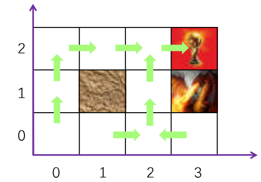
discounted factor, $\gamma \in [0,1)$

$$s_t \xrightarrow[r_t]{a_t} s_{t+1} \xrightarrow[r_{t+1}]{a_{t+1}} s_{t+2} \xrightarrow[r_{t+2}]{a_{t+2}} \cdots$$

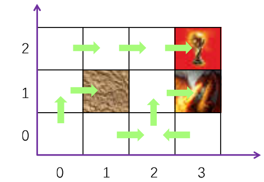
$$a_t = \pi(s_t) \quad r_t = r(s_t, a_t) \quad s_{t+1} = \delta(s_t, a_t)$$

policy reward state transition

Value Function



A good policy π_1



A bad policy π_2

$\gamma = 0.9$

$V^{\pi_1}((0,0)) = 0.9^4 \times 100 = 65.61$
$V^{\pi_1}((1,0)) = 0.9^3 \times 100 = 72.9$
$V^{\pi_1}((2,0)) = 0.9^2 \times 100 = 81.0$
$V^{\pi_1}((3,0)) = 0.9^3 \times 100 = 72.9$
$V^{\pi_1}((0,1)) = 0.9^3 \times 100 = 72.9$
$V^{\pi_1}((2,1)) = 0.9 \times 100 = 90.0$
$V^{\pi_1}((0,2)) = 0.9^2 \times 100 = 81.0$
$V^{\pi_1}((1,2)) = 0.9 \times 100 = 90.0$
$V^{\pi_1}((2,2)) = 100.0$

$V^{\pi_2}((0,0)) = 0$
$V^{\pi_2}((1,0)) = 0.9^2 \times (-100) = -81.0$
$V^{\pi_2}((2,0)) = 0.9 \times (-100) = -90.0$
$V^{\pi_2}((3,0)) = 0.9^2 \times (-100) = -81$
$V^{\pi_2}((0,1)) = 0$
$V^{\pi_2}((2,1)) = -100.0$
$V^{\pi_2}((0,2)) = 0.9^2 \times 100 = 81.0$
$V^{\pi_2}((1,2)) = 0.9 \times 100 = 90.0$
$V^{\pi_2}((2,2)) = 100.0$

Value Function – Bellman Equation

- Starting from an arbitrary state s_t, the cumulative reward by following π is given by

$$V^\pi(s_t) := r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

$$s_t \xrightarrow[r_t]{a_t} s_{t+1} \xrightarrow[r_{t+1}]{a_{t+1}} s_{t+2} \xrightarrow[r_{t+2}]{a_{t+2}} \cdots$$

$$a_t = \pi(s_t) \quad r_t = r(s_t, a_t) \quad s_{t+1} = \delta(s_t, a_t)$$

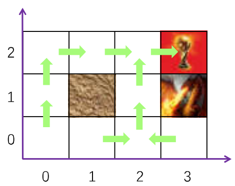
- **Bellman Equation**

$$\begin{aligned} V^\pi(s_t) &= r_t + \gamma\left(r_{t+1} + \gamma r_{t+2} + \ldots\right) \\ &= r_t + \gamma V^\pi(s_{t+1}) \\ &= r_t + \gamma V^\pi(\delta(s_t, a_t)) \end{aligned}$$

Value Function – Bellman Equation

- **Bellman Equation**

$$V^\pi(s) = r(s,a) + \gamma V^\pi(\delta(s,a))$$



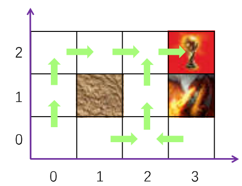
A good policy π_1

$$\begin{pmatrix} V^{\pi_1}((0,0)) \\ V^{\pi_1}((1,0)) \\ V^{\pi_1}((2,0)) \\ V^{\pi_1}((3,0)) \\ V^{\pi_1}((0,1)) \\ V^{\pi_1}((1,1)) \\ V^{\pi_1}((2,1)) \\ V^{\pi_1}((3,1)) \\ V^{\pi_1}((0,2)) \\ V^{\pi_1}((1,2)) \\ V^{\pi_1}((2,2)) \\ V^{\pi_1}((3,2)) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 100 \\ 0 \end{pmatrix} + \gamma \begin{pmatrix} 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} V^{\pi_1}((0,0)) \\ V^{\pi_1}((1,0)) \\ V^{\pi_1}((2,0)) \\ V^{\pi_1}((3,0)) \\ V^{\pi_1}((0,1)) \\ V^{\pi_1}((1,1)) \\ V^{\pi_1}((2,1)) \\ V^{\pi_1}((3,1)) \\ V^{\pi_1}((0,2)) \\ V^{\pi_1}((1,2)) \\ V^{\pi_1}((2,2)) \\ V^{\pi_1}((3,2)) \end{pmatrix}$$

Value Function – Bellman Equation

- **Bellman Equation**

$$V^\pi(s) = r(s,a) + \gamma V^\pi(\delta(s,a))$$



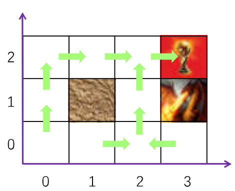
A good policy π_1

$$\begin{pmatrix} V^{\pi_1}((0,0)) \\ V^{\pi_1}((1,0)) \\ V^{\pi_1}((2,0)) \\ V^{\pi_1}((3,0)) \\ V^{\pi_1}((0,1)) \\ V^{\pi_1}((1,1)) \\ V^{\pi_1}((2,1)) \\ V^{\pi_1}((3,1)) \\ V^{\pi_1}((0,2)) \\ V^{\pi_1}((1,2)) \\ V^{\pi_1}((2,2)) \\ V^{\pi_1}((3,2)) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 100 \\ 0 \end{pmatrix} + \gamma \begin{pmatrix} 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} V^{\pi_1}((0,0)) \\ V^{\pi_1}((1,0)) \\ V^{\pi_1}((2,0)) \\ V^{\pi_1}((3,0)) \\ V^{\pi_1}((0,1)) \\ V^{\pi_1}((1,1)) \\ V^{\pi_1}((2,1)) \\ V^{\pi_1}((3,1)) \\ V^{\pi_1}((0,2)) \\ V^{\pi_1}((1,2)) \\ V^{\pi_1}((2,2)) \\ V^{\pi_1}((3,2)) \end{pmatrix}$$

Value Function – Bellman Equation

- **Bellman Equation**

$$V^\pi(s) = r(s,a) + \gamma V^\pi(\delta(s,a))$$



A good policy π_1

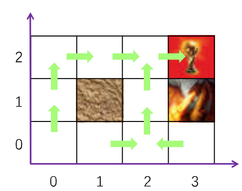
$$\begin{pmatrix} V^{\pi_1}((0,0)) \\ V^{\pi_1}((1,0)) \\ V^{\pi_1}((2,0)) \\ V^{\pi_1}((3,0)) \\ V^{\pi_1}((0,1)) \\ V^{\pi_1}((1,1)) \\ V^{\pi_1}((2,1)) \\ V^{\pi_1}((3,1)) \\ V^{\pi_1}((0,2)) \\ V^{\pi_1}((1,2)) \\ V^{\pi_1}((2,2)) \\ V^{\pi_1}((3,2)) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 100 \\ 0 \end{pmatrix} + \gamma \begin{pmatrix} 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} V^{\pi_1}((0,0)) \\ V^{\pi_1}((1,0)) \\ V^{\pi_1}((2,0)) \\ V^{\pi_1}((3,0)) \\ V^{\pi_1}((0,1)) \\ V^{\pi_1}((1,1)) \\ V^{\pi_1}((2,1)) \\ V^{\pi_1}((3,1)) \\ V^{\pi_1}((0,2)) \\ V^{\pi_1}((1,2)) \\ V^{\pi_1}((2,2)) \\ V^{\pi_1}((3,2)) \end{pmatrix}$$

$$V = R + \gamma T V$$

Value Function – Bellman Equation

- **Bellman Equation**

$$V^\pi(s) = r(s,a) + \gamma V^\pi(\delta(s,a))$$



A good policy π_1

$$V = R + \gamma T V$$

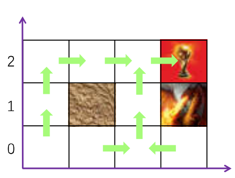
$$\Downarrow$$

$$V = (I - \gamma T)^{-1} R$$

Value Function – Bellman Equation

- **Bellman Equation**

$$V^\pi(s) = r(s,a) + \gamma V^\pi(\delta(s,a))$$



A good policy π_1

$$V = R + \gamma T V$$

$$\Downarrow$$

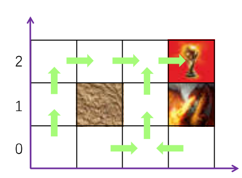
$$V = \boxed{(I - \gamma T)}^{-1} R$$

invertible ?

Value Function – Bellman Equation

- **Bellman Equation**

$$V^\pi(s) = r(s,a) + \gamma V^\pi(\delta(s,a))$$



A good policy π_1

Theorem: *For a finite MDP, Bellman's equation admits a unique solution that is given by*

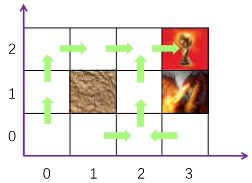
$$V = (I - \gamma T)^{-1} R$$

- The vector R and matrix T depend on the policy

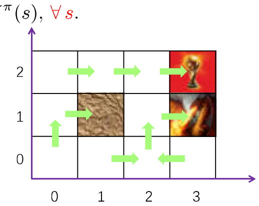
The Learning Task Revisited

- The learning task for RL scenarios is to learn an optimal policy in the sense that

$$\pi^* := \mathrm{argmax}_\pi V^\pi(s),\ \forall s.$$



A good policy π_1



A bad policy π_2

- For π_1 and π_2, we have

$$V^{\pi_1}(s) \geq V^{\pi_2}(s),\ \forall s.$$

- Indeed, π_1 is the optimal policy.

The Q Function

- Learning the optimal policy is challenging
- An alternative approach to find the optimal policy indirectly is by computing the state-action value function (Q function)

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

$Q(s,a)$ is the accumulated reward by performing the action a first and then following the optimal policy

- The definition of the optimal policy implies that

$$\pi^*(s) = \mathrm{argmax}_a Q(s,a) = \mathrm{argmax}_a r(s,a) + \gamma V^*(\delta(s,a))$$

- Notice that

$$V^*(s) = \max_a Q(s,a) = \max_a r(s,a) + \gamma V^*(\delta(s,a))$$

- All together, we have

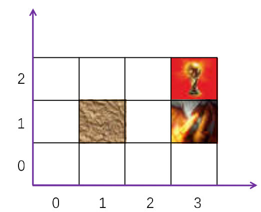
$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a),a')$$

Bellman Equations

Planning Algorithms

Planning

- Planning: to find the optimal policy, there is no need for the agent to actually perform actions and interact with the environment



Known
$\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$: state transition
$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: reward

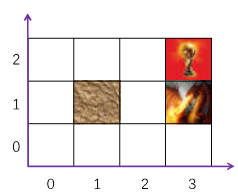
Value Iteration

- Value iteration aims to find the optimal value function and thus the optimal policy

Initialize $V(s)$ to arbitrary values
while termination conditions does not hold
  For $s \in \mathcal{S}$
    For $a \in \mathcal{A}$
      $Q(s,a) \leftarrow r(s,a) + \gamma V(\delta(s,a))$
    $V(s) \leftarrow \max_a Q(s,a)$

Value Iteration

- Value iteration aims to find the optimal value function and thus the optimal policy



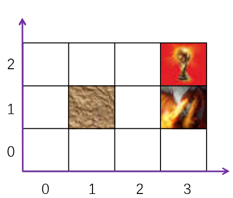
Example

$V \leftarrow 0$
$Q((0,0),\mathrm{up}) \leftarrow 0 + 0.9 \times V((0,1)) = 0$
$Q((0,0),\mathrm{down}) \leftarrow 0 + 0.9 \times V((0,0)) = 0$
$Q((0,0),\mathrm{left}) \leftarrow 0 + 0.9 \times V((0,0)) = 0$
$Q((0,0),\mathrm{right}) \leftarrow 0 + 0.9 \times V((1,0)) = 0$
$V((0,0)) \leftarrow \max\{Q((0,0),\mathrm{up}), Q((0,0),\mathrm{down}), Q((0,0),\mathrm{left}), Q((0,0),\mathrm{right})\} = 0$

Nothing happens

Value Iteration

- Value iteration aims to find the optimal value function and thus the optimal policy



Example

$V \leftarrow 0$
$Q((2,2),\mathrm{up}) \leftarrow 0 + 0.9 \times V((2,2)) = 0$
$Q((2,2),\mathrm{down}) \leftarrow 0 + 0.9 \times V((2,1)) = 0$
$Q((2,2),\mathrm{left}) \leftarrow 0 + 0.9 \times V((1,2)) = 0$
$Q((2,2),\mathrm{right}) \leftarrow 100 + 0.9 \times V((3,2)) = 100$
$V((2,2)) \leftarrow \max\{Q((2,2),\mathrm{up}), Q((2,2),\mathrm{down}), Q((2,2),\mathrm{left}), Q((2,2),\mathrm{right})\} = 100$

Value Iteration

- Value iteration aims to find the optimal value function and thus the optimal policy

Theorem: *For any initial value V, the sequence generated by the value iteration algorithm converges to V^*.*

- The key to the proof is the contraction mapping theorem

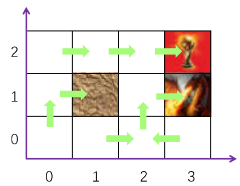
Policy Iteration

- Policy iteration improves the policy directly

$$
\begin{aligned}
&\textbf{Initialize } \pi, \pi' \text{ to two different policies}\\
&\textbf{while}(\pi \neq \pi')\\
&\quad V \leftarrow (I - \gamma T^{\pi})^{-1} R^{\pi}\\
&\quad \pi' \leftarrow \pi\\
&\qquad \textbf{For } s \in \mathcal{S}\\
&\qquad \pi(s) \leftarrow \operatorname{argmax}_a r(s,a) + \gamma V(\delta(s,a))
\end{aligned}
$$

Policy Iteration

- Policy iteration improves the policy directly

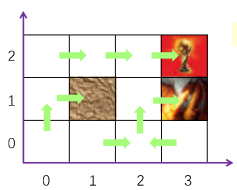


A bad policy π_2

$$
\begin{aligned}
&\textbf{Initialize } \pi \leftarrow \pi_2, \pi' \neq \pi_2\\
&\textbf{while}(\pi \neq \pi')\\
1^{st} &\quad V \leftarrow (I - \gamma T^{\pi})^{-1} R^{\pi}\\
&\quad \pi' \leftarrow \pi\\
&\qquad \textbf{For } s \in \mathcal{S}\\
&\qquad \pi(s) \leftarrow \operatorname{argmax}_a r(s,a) + \gamma V(\delta(s,a))
\end{aligned}
$$

Policy Iteration

- Policy iteration improves the policy directly



A bad policy π_2

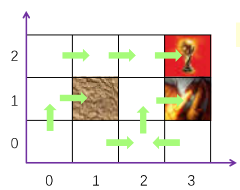
1^{st} iteration

$$
V^{\pi} = \begin{pmatrix} V^{\pi}(0,0)\\ V^{\pi}(1,0)\\ V^{\pi}(2,0)\\ V^{\pi}(3,0)\\ V^{\pi}(0,1)\\ V^{\pi}(2,1)\\ V^{\pi}(3,1)\\ V^{\pi}(0,2)\\ V^{\pi}(1,2)\\ V^{\pi}(2,2)\\ V^{\pi}(3,2) \end{pmatrix}
$$

11 states in total

Policy Iteration

- Policy iteration improves the policy directly



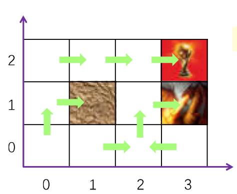
A bad policy π_2

1^{st} iteration

$$
R^{\pi} = \begin{pmatrix} r((0,0),\text{up})\\ r((1,0),\text{right})\\ r((2,0),\text{up})\\ r((3,0),\text{left})\\ r((0,1),\text{right})\\ r((2,1),\text{right})\\ r((3,1),\text{END})\\ r((0,2),\text{right})\\ r((1,2),\text{right})\\ r((2,2),\text{right})\\ r((3,2),\text{END}) \end{pmatrix} = \begin{pmatrix} 0\\0\\0\\0\\0\\-100\\0\\0\\0\\100\\0 \end{pmatrix}
$$

Policy Iteration

- Policy iteration improves the policy directly



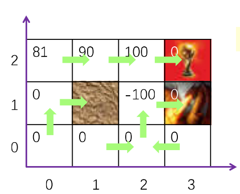
A bad policy π_2

1^{st} iteration

$$
T^{\pi} = \begin{pmatrix}
0&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}
$$

Policy Iteration

- Policy iteration improves the policy directly



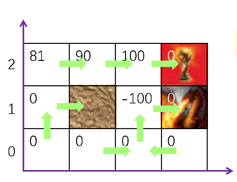
A bad policy π_2

1^{st} iteration

$$
V \leftarrow (I - \gamma T^{\pi})^{-1} R^{\pi} = \begin{pmatrix} 0\\0\\0\\0\\0\\-100\\0\\81\\90\\100\\0 \end{pmatrix}
$$

Policy Iteration

- Policy iteration improves the policy directly



A bad policy π_2

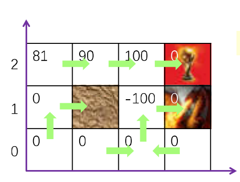
1^{st} iteration: update the policy

$$
\begin{aligned}
\pi((0,0)) = \operatorname{argmax}_a \{& r((0,0),\text{up}) + \gamma V((0,1)),\\
& r((0,0),\text{down}) + \gamma V((0,0)),\\
& r((0,0),\text{left}) + \gamma V((0,0)),\\
& r((0,0),\text{right}) + \gamma V((1,0))\}\\
= \operatorname{argmax}_a & \{0,0,0,0\}
\end{aligned}
$$

We can randomly select one action from $\mathcal{A} = \{\text{up, down, left, right}\}$. However, it is better select one action from up and right (why?).

Policy Iteration

- Policy iteration improves the policy directly



A bad policy π_2

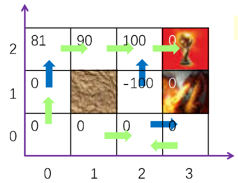
1^{st} iteration: update the policy

$$
\begin{aligned}
\pi((0,0)) = \operatorname{argmax}_a \{& r((0,0),\text{up}) + \gamma V((0,1)),\\
& r((0,0),\text{down}) + \gamma V((0,0)),\\
& r((0,0),\text{left}) + \gamma V((0,0)),\\
& r((0,0),\text{right}) + \gamma V((1,0))\}\\
= \operatorname{argmax}_a & \{0,0,0,0\}
\end{aligned}
$$

We can indeed assign negative rewards for actions that will not alter the states when these states are not the goal states. Or, we can simply ignore these actions.

Policy Iteration

- Policy iteration improves the policy directly



A bad policy π_2

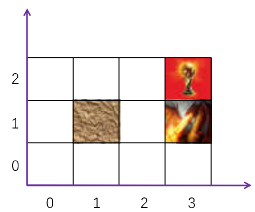
1^{st} iteration: update the policy

$\pi((0,0)) = \text{up}$
$\pi((1,0)) = \text{right}$
$\pi((2,0)) = \text{right}$
$\pi((3,0)) = \text{left}$
$\pi((0,1)) = \text{up}$
$\pi((2,1)) = \text{up}$
$\pi((3,1)) = \text{END}$
$\pi((0,2)) = \text{right}$
$\pi((1,2)) = \text{right}$
$\pi((2,2)) = \text{right}$
$\pi((3,2)) = \text{END}$

Learning Algorithms

Learning

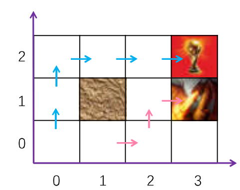
- Learning: as the environment model, i.e., the transition and reward, is unknown, the agent may need to learn them based on the training information.



Unknown
$\delta : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$: state transition
$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$: reward

Learning

- Learning: as the environment model, i.e., the transition and reward, is unknown, the agent may need to learn them based on the training information.
  - Model-free approach: the agent learns the optimal policy directly, e.g., Q-learning
  - Model-based approach: the agent first learns the environment model and then the optimal policy



Examples of training data

$(0,0) \xrightarrow[0]{\text{up}} (0,1) \xrightarrow[0]{\text{up}} (0,2) \xrightarrow[0]{\text{right}} (1,2) \xrightarrow[0]{\text{right}} (2,3) \xrightarrow[100]{\text{right}} (3,2)$

$(1,0) \xrightarrow[0]{\text{right}} (2,0) \xrightarrow[0]{\text{up}} (2,1) \xrightarrow[-100]{\text{right}} (3,1)$

The Q-learning Algorithm

- Initialize the matrix $\hat{Q}$ to zero
- Observe the current state s
- Do forever:
  - Pick and perform an action a
  - Receive immediate reward r
  - Observe the new state s'
  - Update

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

  - $s \leftarrow s'$

A sufficient condition for $\hat{Q}(s,a)$ to converge is to visit each state-action pair infinitely often

The Q-learning Algorithm

- Initialize the matrix $\hat{Q}$ to zero
- Observe the current state s
- Do forever:
  - Pick and perform an action a
  - Receive immediate reward r
  - Observe the new state s'
  - Update

How to pick the action?

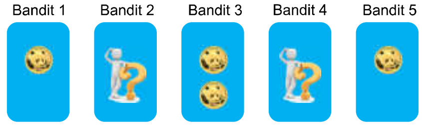
$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

  - $s \leftarrow s'$

Exploitation vs Exploration

- Multi-armed bandit

Bandit 1 Bandit 2 Bandit 3 Bandit 4 Bandit 5

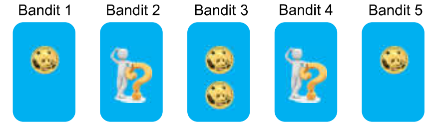


- Which machine next?
  - Exploitation: the machine with the largest reward at present
  - Exploration: randomly select a machine

Exploitation vs Exploration

- Multi-armed bandit

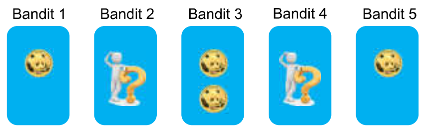
Bandit 1 Bandit 2 Bandit 3 Bandit 4 Bandit 5



- ϵ-greedy
  - with probability $1-\epsilon$, we do exploitation
  - with probability ϵ, we do exploration, i.e., we uniformly randomly select an action from all possible actions
- Tips for ϵ-greedy
  - At the beginning, the agent does not know the environment very well. Thus, it need to do more exploration and a large value of ϵ is needed.
  - When the environment model is well explored, the agent can do more exploitation. Thus, we favor a small value of ϵ.

Exploitation vs Exploration

- Multi-armed bandit

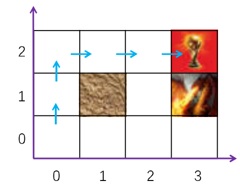


- A soft sampling strategy
  - Given a state, we can choose action probabilistically

$$\mathrm{P}[a|s] = \frac{e^{\hat{Q}(s,a)/T}}{\sum_{a'} e^{\hat{Q}(s,a')/T}}$$

  - Smaller values of T will assign higher probabilities for actions with high $\hat{Q}$, leading to an exploitation strategy.
  - Larger values of T will encourage the agent to explore actions that do not currently have high $\hat{Q}$ values.

The Q-learning Algorithm

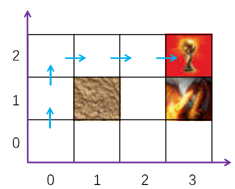


$(0,0) \xrightarrow[0]{\text{up}} (0,1) \xrightarrow[0]{\text{up}} (0,2) \xrightarrow[0]{\text{right}} (1,2) \xrightarrow[0]{\text{right}} (2,3) \xrightarrow[100]{\text{right}} (3,2)$

- an example episode
- the initial state in each episode could NOT be fixed (why?)

$\epsilon = 0.3$

The Q-learning Algorithm



$(0,0) \xrightarrow[0]{\text{up}} (0,1) \xrightarrow[0]{\text{up}} (0,2) \xrightarrow[0]{\text{right}} (1,2) \xrightarrow[0]{\text{right}} (2,3) \xrightarrow[100]{\text{right}} (3,2)$

$\leftarrow \quad \leftarrow \quad \leftarrow \quad \leftarrow \quad \leftarrow$

$\epsilon = 0.3$

Questions