Reinforcement Learning

Introduction to Artificial Intelligence

Yang Gao, Francesca Toni

Russel & Norving, *AI: a modern approach*, Section 21
Sutton & Barto, *Reinforcement Learning: An introduction*

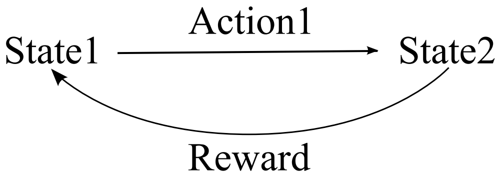
A short history of Reinforcement Learning:

- Proposed by the behavioural psychologists
- Used to train animals (including human beings)
- An area of Machine Learning

RL in a nutshell

In each state:

- if an agent does something “good”, it gets rewards;
- if an agent does something “wrong”, it gets punishments;
- rewards and punishments are received right after the effect of the performed action is observed;
- goal: choose the best action at each state to maximise the long-term rewards.

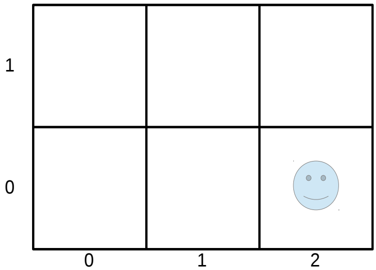


Components of Reinforcement Learning

- **State**: All information needed for choosing actions
- **Action**: Available actions in each state
- **Dynamic of the Environment**: How the environment is changed by the actions
- **Rewards**: Criteria used to evaluate the the goodness of performing an action in a state (typically a real number)

A motivating example: Maze

- A 2×3 grid world
- A pit, an exit and some walls are known in this grid world, but their locations are unknown
- Arrive at the exit: win; fall in the pit: die; hit a wall, hurt
- Goal: Get out of this maze (i.e. safely arrive at the exit) as quickly as possible



RL components in this problem

- **State**: The agent's current location
- **Action**: *LEFT, RIGHT, UP, DOWN*
- **Environment Dynamics**:
  - If in the intended direction there exists a wall, no-op
  - otherwise, move one square in the intended direction
- **Rewards**:
  - normal move: -1
  - hit a wall: -10
  - die: -100
  - exit: +100
- **Our Goal**: find the best route to exit

Markov Decision Process (MDP)

Components of RL (t:current time slot, $t+1$: next time slot):

- **States**: describe the environment;
- **Actions**: available actions for execution;
- **Transition Probability**: $Pr\{s_{t+1}|s_t, a_t\}$ (denoted $\mathcal{P}^{a_t}_{s_t s_{t+1}}$);
- **Rewards**: $E\{r_{t+1}|s_t, a_t, s_{t+1}\}$ (denoted $\mathcal{R}^{a_t}_{s_t s_{t+1}}$);
- $MDP = < S, A, T, R >$.

Why MDP?

- RL problems satisfy the *Markov property*: history independence.

How to find the best action

The goal of RL:

- Formally: in a state $s_0 \in S$, determine the best *policy* π so as to maximise the *long-term reward*
- A *policy* $\pi : S \to A$, is a function from state to action. $\pi(s) = a$ means choose action a at state s
- The long-time reward of performing a_0 at s_0:

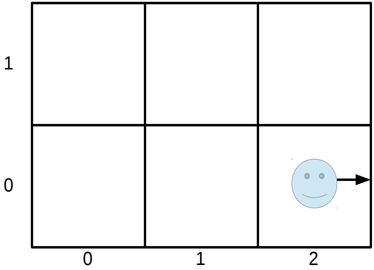
$$Q^\pi(s_0, a_0) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} | s_0, a_0\}$$

where E_π denotes the expected value given that the agent follows policy π, and $\gamma \in [0, 1]$ is the *discount factor* indicating how 'short-sighted' the agent is: the smaller γ, the more short-sighted the agent.

An example policy and its Q-value

- A policy: always go right
- Formally: $\pi(s) = RIGHT$ for all $s \in S$

$$
\begin{aligned}
&Q^{\pi}([0,2], RIGHT) \\
&= -10 - 10 \times \gamma - 10 \times \gamma^2 \cdots \\
&= -10 + \gamma(-10 - 10\gamma - 10\gamma^2 - \cdots) \\
&= -10 + \gamma Q^{\pi}([0,2], RIGHT) \\
&= -10/(1-\gamma)
\end{aligned}
$$



Q-value in recursive form (Bellman Equation)

$$
\begin{aligned}
Q^{\pi}(s_0, a_0) &= E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} | s_0, a_0\} \\
&= E_{\pi}\{r_1 + \gamma \sum_{k=0}^{\infty} \gamma^k r_{k+2} | s_0, a_0\} \\
&= \sum_{s_1 \in S} \mathcal{P}_{s_0 s_1}^{a_0} (\mathcal{R}_{s_0 s_1}^{a_0} + \gamma \sum_{a_1 \in A} \pi(s_1, a_1) \cdot E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{1+k+1} | s_1, a_1\}) \\
&= \sum_{s_1 \in S} \mathcal{P}_{s_0 s_1}^{a_0} (\mathcal{R}_{s_0 s_1}^{a_0} + \gamma \sum_{a_1 \in A} \pi(s_1, a_1) \cdot Q^{\pi}(s_1, a_1))
\end{aligned}
$$

Find the best policy

- Optimal policy: $\pi^* = \arg\max_{\pi} Q^{\pi}(s, a)$ for all $s \in S$, $a \in A$
- Optimal policies share the same Q-values:

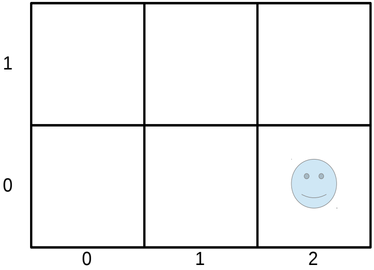
$$
\begin{aligned}
Q^*(s_0, a_0) &= \max_{\pi} Q^{\pi}(s_0, a_0) \\
&= \sum_{s_1 \in S} \mathcal{P}_{s_0 s_1}^{a_0} \left(\mathcal{R}_{s_0 s_1}^{a_0} + \gamma \max_{a_1 \in A} Q^*(s_1, a_1)\right)
\end{aligned}
$$

Q-Learning Algorithm

Initialise $Q(s, a)$ for all states s and all actions a arbitrarily
Repeat (for each episode):
    Initialise s
    Repeat (for each step in an episode):
        Choose $a = \arg\max_{a \in A} Q(s, a)$ with probability $1 - \epsilon$
        Perform action a, observe reward r and next state s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    until s is terminal

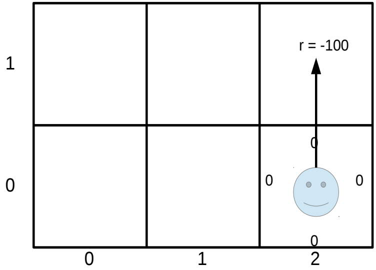
Applying Q-Learning to The Maze Problem

- $\alpha = 0.5$, $\gamma = 0.9$
- All Q-values are initialised as 0



Applying Q-Learning to The Maze Problem

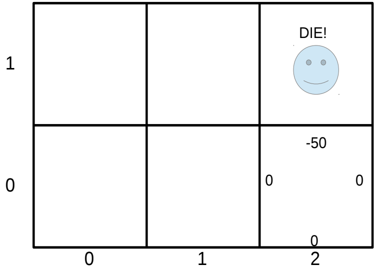
- $\alpha = 0.5$, $\gamma = 0.9$
- All Q-values are initialised as 0
- Choose *UP*, and receive -100



Applying Q-Learning to The Maze Problem

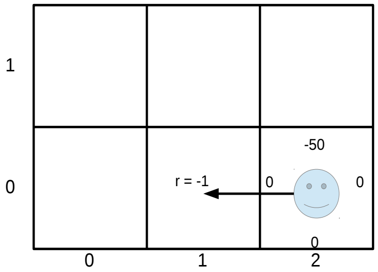
- $\alpha = 0.5$, $\gamma = 0.9$
- All Q-values are initialised as 0
- Choose *UP*, and receive -100
- update Q-value:

$$Q([0,2], UP) \\ = (1 - 0.5) \times 0 + \\ 0.5 \times (-100 + 0.9 \times 0) \\ = -50$$



Applying Q-Learning to The Maze Problem

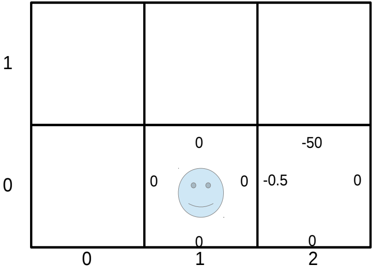
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1



Applying Q-Learning to The Maze Problem

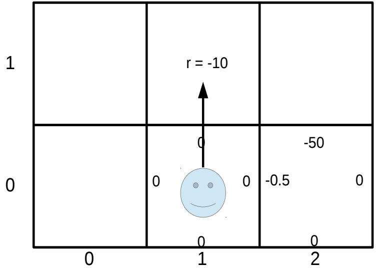
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1
- update Q-value:

$$Q([0,2], LEFT) \\ = (1 - 0.5) \times 0 + \\ 0.5 \times (-1 + 0.9 \times 0) \\ = -0.5$$



Applying Q-Learning to The Maze Problem

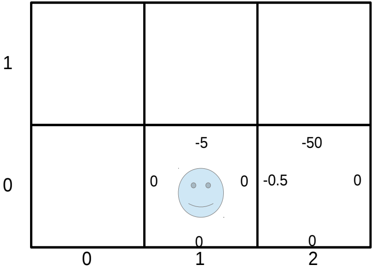
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -10



Applying Q-Learning to The Maze Problem

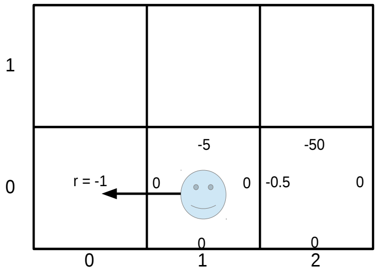
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -10
- update Q-value:

$$Q([0,1], UP)$$
$$= (1-0.5)\times 0 +$$
$$0.5 \times (-10 + 0.9 \times 0)$$
$$= -5$$



Applying Q-Learning to The Maze Problem

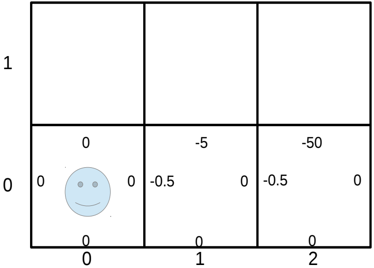
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1



Applying Q-Learning to The Maze Problem

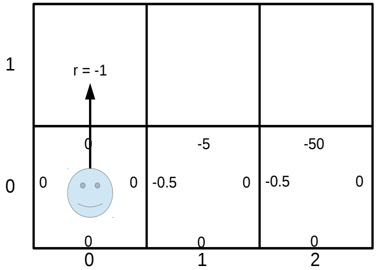
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *LEFT*, and receive -1
- update Q-value:

$$Q([0,1], LEFT)$$
$$= (1-0.5)\times 0 +$$
$$0.5 \times (-1 + 0.9 \times 0)$$
$$= -0.5$$



Applying Q-Learning to The Maze Problem

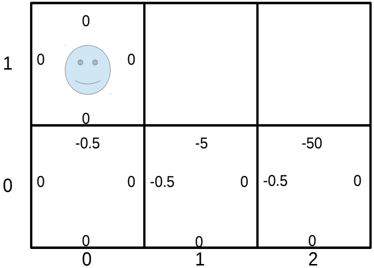
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -1



Applying Q-Learning to The Maze Problem

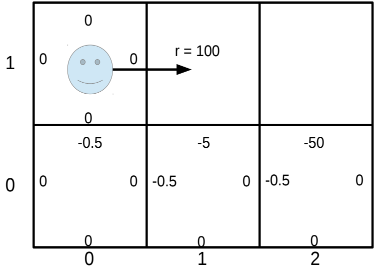
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *UP*, and receive -1
- update Q-value:

$$Q([0,0], UP)$$
$$= (1 - 0.5) \times 0 +$$
$$0.5 \times (-1 + 0.9 \times 0)$$
$$= -0.5$$



Applying Q-Learning to The Maze Problem

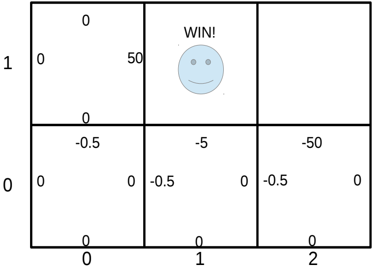
- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *RIGHT*, and receive 100



Applying Q-Learning to The Maze Problem

- $\alpha = 0.5$, $\gamma = 0.9$
- Choose *RIGHT*, and receive 100
- update Q-value:

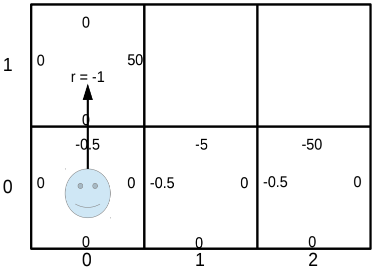
$$Q([0,1], RIGHT) \\ = (1 - 0.5) \times 0 + \\ 0.5 \times (100 + 0.9 \times 0) \\ = 50$$



Applying Q-Learning to The Maze Problem

- $\alpha = 0.5$, $\gamma = 0.9$
- The next time agent visits [0,0] and performs *UP*:

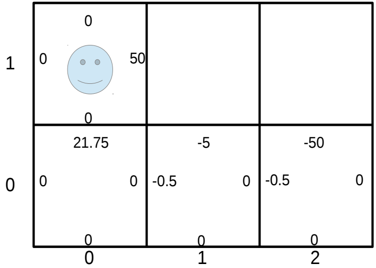
$$Q([0,0], UP)$$
$$= (1 - 0.5) \times (-0.5) +$$
$$0.5 \times (-1 + 0.9 \times 50)$$
$$= 21.75$$



Applying Q-Learning to The Maze Problem

- $\alpha = 0.5$, $\gamma = 0.9$
- The next time agent visits [0,0] and performs *UP*:

$$Q([0,0], UP)$$
$$= (1 - 0.5) \times (-0.5) + 0.5 \times (-1 + 0.9 \times 50)$$
$$= 21.75$$



Property of Q-Learning

- Quick learning speed: Q-Learning is able to find a quite good policy after a small number of iterations.
- Model-free: no need to keep record of the learning trajectory (no need to explicitly compute the Transition Probability).
- Guarantee to converge: after visiting each state-action pairs for an infinite number of times, for any $s \in S, a \in A$, $Q(s, a)$ is guaranteed to converge to $Q^*(s, a)$.

The learning parameters in Q-Learing

- α:
  - Learning step
  - balance between existing experiences (weight: $1 - \alpha$) and new observations (weight: α)
- γ:
  - Future discount
  - balance between current reward (weight: 1) and next N step's reward (weight: γ^N)
- ϵ:
  - indicating how 'bold' the agent is
  - balance between *exploitation* (take greedy action, $1 - \epsilon$ chance) and *exploration* (take random action, ϵ chance)

Summary

- Basic idea of Reinforcement Learning (RL):
  - did something "good" — reward
  - did something "bad" — punish
- Formal model of RL — Markov Decision Process (MDP):
  - $MDP =< State, Action, TransProb, Reward >$
  - $Q(s, a)$: to valuate the value of performing action a in state s
  - Optimal policy:
    - $\pi^* = \arg\max_\pi Q^\pi$
    - $Q^{\pi^*} = Q^*$
  - Q-Learning algorithm

What makes RL different?

In which problems we should use RL instead of other ML techniques?

- Sequential decision making: time and order really matters
- I know what I want, but don't know how to get it: no supervisor, only rewards (goals)
- The agent can interact with the environment: trial and error is affordable
- Feedback is delayed: only after observing actions' consequences

Tutorial: The Wumpus World

A *Wumpus World* example:

- A 5×5 grid world
- The world has: an exit, some golds, some Wumpus and some pits. The agent only knows the golds' locations
- The agent can shoot a Wumpus next to it; by killing a Wumpus, the agent gets reward
- When next to a Wumpus: smells *stench*;
  next to a pit: feels *breeze*;
  on a gold: sees *glitter*
- Step into a square with a Wumpus or a pit: die;
  hit wall: no-op
- Task: collects all gold and arrives the exit safely

Tutorial: The Wumpus World

- Actions:
  LEFT, *RIGHT*, *UP*, *DOWN*,
  SHOOT_LEFT, *SHOOT_RIGHT*, *SHOOT_UP*,
  SHOOT_DOWN,
  PICK_UP
- Rewards:
  - Normal move: -1
  - Die: -1000
  - Get a gold or kill a Wumpus: +100
  - Missed shoot or missed pick: -50
  - Successfully exit the world: +500

RL for Wumpus World: Understand The Learning Parameters

- Default parameters:
  - $\alpha = 0.1$
  - $\gamma = 0.9$
  - $\epsilon = 0.05$
- Run the following experiments to see how each parameter affects the learning performance (100 episodes for each)
  - Keeping the default value for γ and ϵ, compare the performances when $\alpha = 0, 0.1, 0.5, 1.0$
  - Keeping the default value for α and ϵ, compare the performances when $\gamma = 0, 0.1, 0.5, 1.0$
  - Keeping the default value for α and γ, compare the performances when $\epsilon = 0, 0.1, 0.5, 1.0$

Domain Knowledge for The Wumpus World

- In certain situations, the goodness of some actions can be deduced by the goodness (Q-value) of other actions
- Suppose an agent steps into a square where:
  - In previous episodes, UP, LEFT, DOWN have been performed in this square, and they are all OK; and
  - the agent can smell stench in this square.
- In this situation, we can easily see that there must exist a Wumpus in the square on the right.
- However, classical RL algorithms cannot deduce this: it still needs to perform RIGHT to realise that there is a Wumpus in the right square

Add Arguments to RL

- We describe the domain knowledge by *Arguments*: logic sentences describing which actions are recommended and/or discouraged.
- The argument for the example domain knowledge above:

IF

left	OK
up	OK
down	OK
stench	TRUE

THEN

$+$	SHOOT_RIGHT
$-$	RIGHT

DONE