Reinforcement Learning

Peerapon S.

Topics

Examples and problem formulation

Policy and value function

Off-line learning

Q-learning

Temporal difference learning

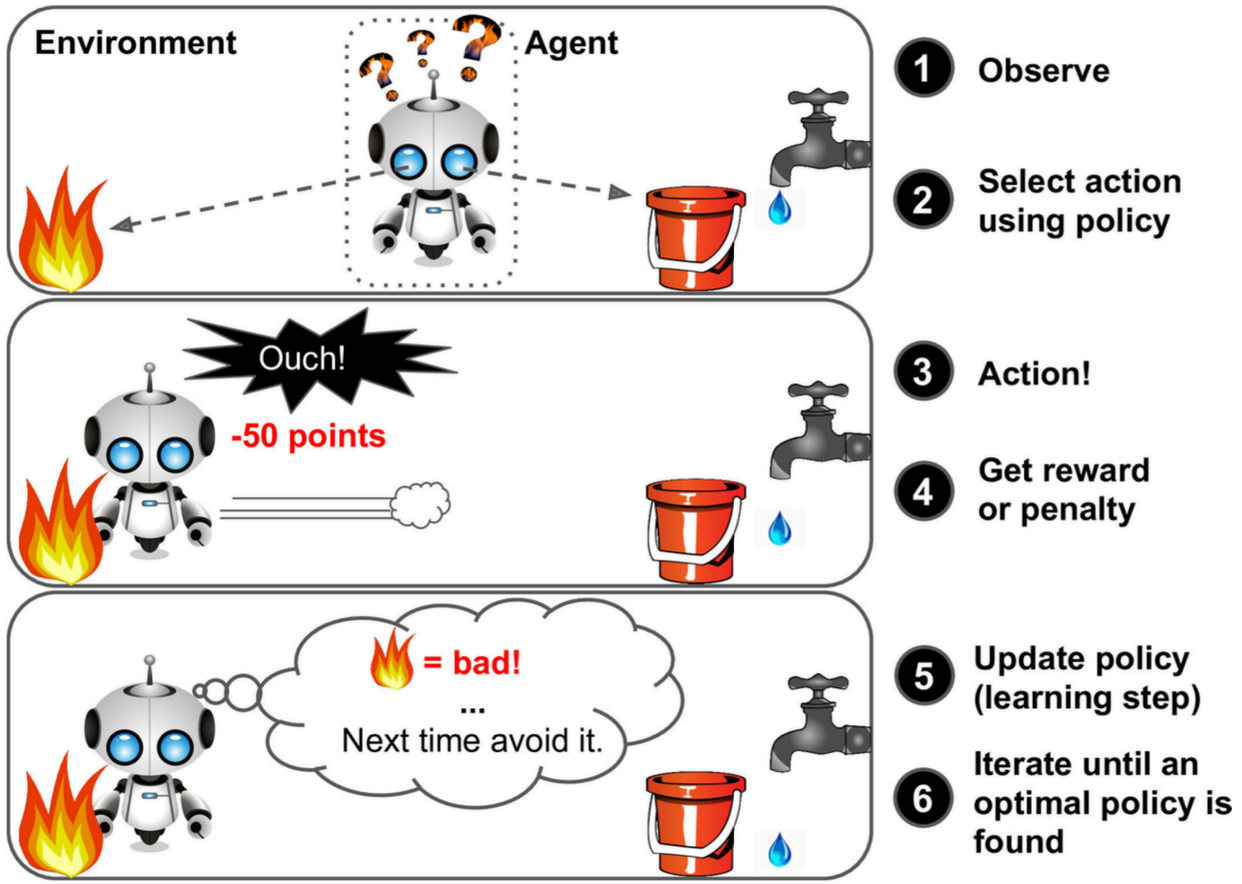
ANOTHER VIEW OF MACHINE LEARNING

TEACHING THE COMPUTER TO LEARN FROM EXPERIENCES AND OPTIMIZE A GIVEN PERFORMANCE INDEX AS THEY PRACTICE.

ATARI Game

Running Bot

Basic Concept



A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras & Tensor Flow, 2019

Problem Formulation

Agent (Decision maker) interacts with ***environment*** to complete a task.

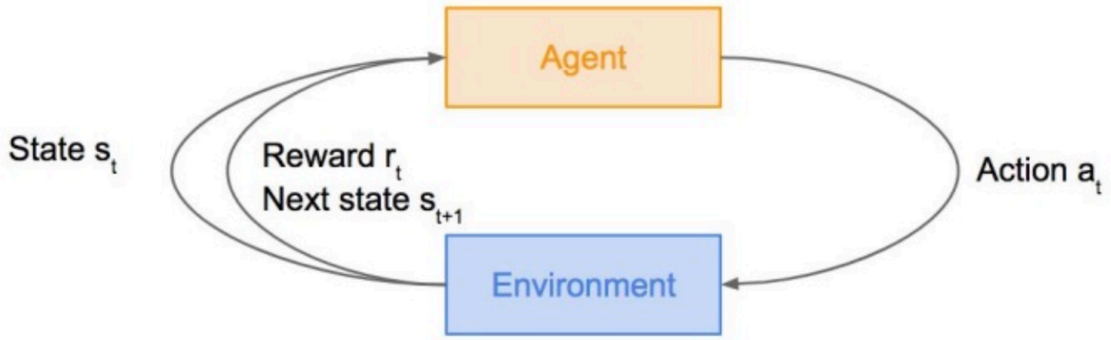
At any point in time, the agent

- Observes the current ***state*** $s_t \in S$ of the environment.
- Selects an ***action*** $a_t \in A$ to perform, resulting in change of state to s_{t+1}.
- Receives an immediate ***reward*** (feedback) r_t from the environment.

Each attempt at the task is referred to as an ***episode*** or ***trial***.

Goal: Complete the task as successfully as possible -- maximizing the ***cumulative reward across an episode***.

$$(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \ \ldots \ , (s_t, a_t, r_t), \ \ldots$$



The ***reward function*** $r(\cdot)$ and ***state transition function*** $\delta(\cdot)$ are parts of the environment, referred to as ***model of the world/environment***.

Model possibilities:

- Known/Not known to the agent
- Independent or dependent on past states and actions
- Deterministic or Non-deterministic (Stochastic)

Ex: Atari's Arcade Game

State = Positions of remaining raw pixels

Action = Game controls - Left, Right

Reward = Score increase/decrease at each time step

Task = Complete the game with least number of actions.

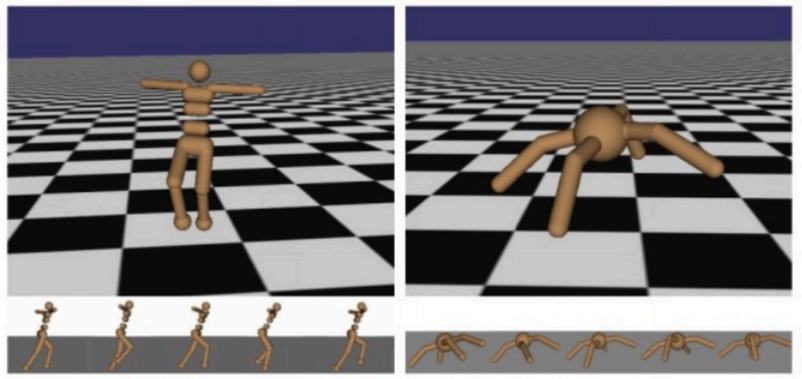
Ex: Robot Locomotion

State = Leg/body positions

Action = Leg/body movement

Reward = +1 if not falling down, 0 otherwise

Task = Make the robot move forward without falling down to the destination.

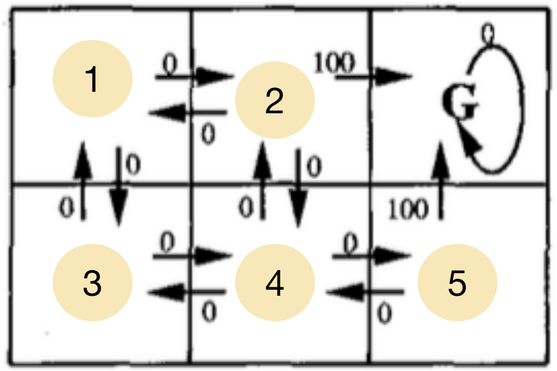


Ex: Six-grid-squares Environment

Assume $r(\cdot)$ and $\delta(\cdot)$ being deterministic and independent from past states and actions so that $r_t = r(s_t, a_t)$ and $s_{t+1} = \delta(s_t, a_t)$.

- What is the set of states ? {1, 2, 3, 4, 5, G}
- What is the set of actions ? {Left, Right, Top, Down, Self}
- What is the reward ? {0, 100}

Task = Go the cell G (Goal/Terminal/Absorbing state) with least number of moves from any initial cell.

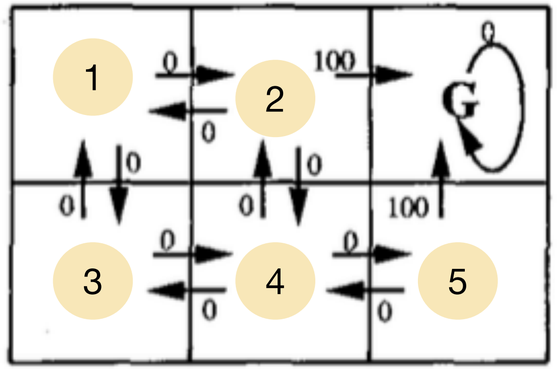


Immediate reward values $r(a, s)$

Policy

A policy (function) $\pi : S \to A$ is a ***mapping*** from $s_t \in S$ to $a_t \in A$, i.e., what action $a_t = \pi(s_t)$ to take in each state s_t.

Given $\pi(s)$, the action in state s is completely determined by s.



Ex: ***Greedy action selection policy***

- Always take the action yielding the highest immediate reward.
- Fail to handle different characteristics of rewards, e.g., ***delayed*** or ***contradictory***.

Need a different measure to decide an action in a given state.

Value Function -- Discounted Reward

Defined as a *cumulative reward* expected to earn when the agent is in state s_t and follows the policy π.

For $0 \leq \gamma < 1$, the ***discounted (cumulative) reward*** obtained from using the policy π is defined by

discount factor

$$V^{\pi}(s_t) \triangleq r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

$$= r_t + \gamma (r_{t+1} + \gamma r_{t+2} + \ldots)$$

$$= r_t + \gamma \cdot V^{\pi}(s_{t+1})$$

$V^{\pi}(s)$ has the recurrence relation with that of next state s'.

What is the implication of discount factor γ ?

discount factor สูง จะให้ความสำคัญกับ future reward

Optimal Policy

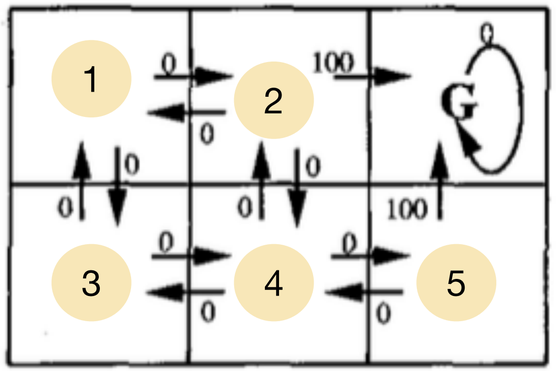
The goal of RL is the agent learns the optimal policy -- Maximizing the cumulative reward in the long run.

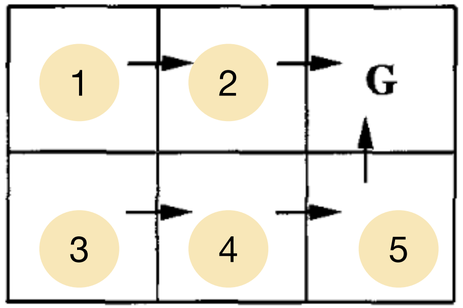
The optimal policy is defined as $\pi^*(s) \triangleq \arg\max_{\pi} V^{\pi}(s), \forall s$

with the optimal policy value $V^{\pi^*}(s) \triangleq V^*(s)$

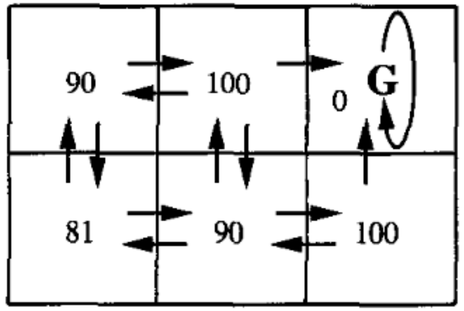
So, the optimal policy always yields the highest policy value for any starting state.

Example of Optimal Policy





One optimal policy



$V^*(s)$ values

Optimal Policy Value

The value of the optimal policy is

$$V^*(s_t) = \max_{a_t, a_{t+1}, a_{t+2}, \ldots} \left\{ r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \cdots \right\}$$

Since the immediate reward $r_t = r(s_t, a_t)$ only depends on s_t and a_t

$$\begin{aligned} V^*(s_t) &= \max_{a_t} [r(s_t, a_t) + \gamma \max_{a_{t+1}, a_{t+2}, \ldots} (r(s_{t+1}, a_{t+1}) + \gamma\, r(s_{t+2}, a_{t+2}) + \cdots)] \\ &= \max_{a_t} [r(s_t, a_t) + \gamma\, V^*(s_{t+1})] \\ \boxed{V^*(s) = \max_a [r(s, a) + \gamma\, V^*(\delta(s, a))]} & \qquad (1) \end{aligned}$$

Recurrent relation

If $r(s, a)$ and $\delta(s, a)$ are known beforehand, $V^*(s)$ can be iteratively approximated ***off-line***, known as ***off-line learning*** or ***model-based learning***.

Off-line Learning: Value Iteration Approximation

For known $r(\cdot)$ and $\delta(\cdot)$, $V^*(s)$ can be iteratively approximated by using

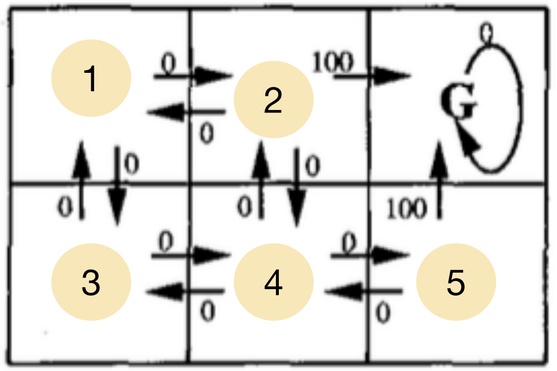
$$\hat{V}(s) \leftarrow \max_a [r(s,a) + \gamma \hat{V}(\delta(s,a))] \qquad (2)$$

ให้ reward เป็น max score

Stop when $\hat{V}(s)$ converges.

From (1), once $V^*(s)$ has been determined, the optimal policy at each state s is obtained from

$$\pi^*(s) = \arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))] \qquad (3)$$



$$\hat{V}(s) \leftarrow \max_a [r(s,a) + \gamma \hat{V}(\delta(s,a))]$$

Initial

s	1	2	3	4	5	G
$V(s)$	0	0	0	0	0	0

1st Iteration:

V(1) = max(0 + 0.9 × 0, 0 + 0.9 × 0) = 0

V(2) = max(0 + 0.9 × 0, 0 + 0.9 × 0, 100 + 0.9 × 0) = 100

converges

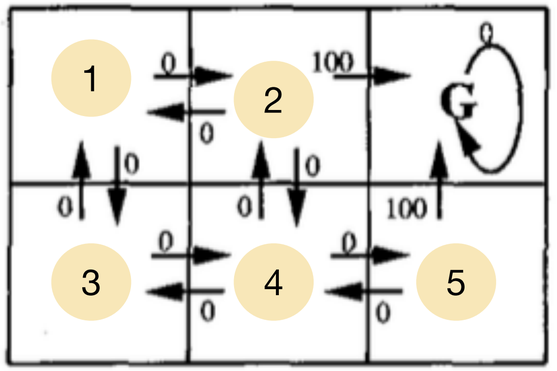
V(3) = 0

V(4) = 0

V(5) = 100

V(G) = 0

s	1	2	3	4	5	G
$V(s)$	0	100	0	0	100	0



สถานะที่เดินไปหา

$$\hat{V}(s) \leftarrow \max_{a}[r(s,a) + \gamma \hat{V}(\delta(s,a))]$$

s	1	2	3	4	5	G
$V(s)$	0	100	0	0	100	0

2nd Iteration:

V(1) = max(0 + 0.9 × 0, 0 + 0.9 × 100) = 90

V(2) = 100

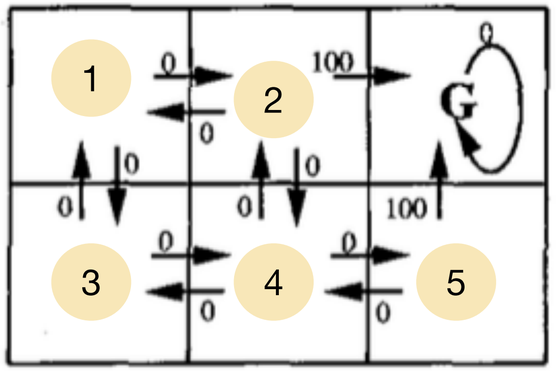
V(3) = max(0 + 0.9 × 90, 0 + 0.9 × 0) = 81

V(4) = max(0 + 0.9(81), 0 + 0.9(100), 0 + 0.9(100)) = 90

V(5) = 100

V(G) = 0

s	1	2	3	4	5	G
$V(s)$	90	100	81	90	100	0



$$\hat{V}(s) \leftarrow \max_a [r(s,a) + \gamma \hat{V}(\delta(s,a))]$$

discount factor

s	1	2	3	4	5	G
$V(s)$	90	100	81	90	100	0

3rd Iteration:

V(1) = 90

V(2) = 100

V(3) = 81

V(4) = 90

V(5) = 100

V(G) = 0

s	1	2	3	4	5	G
$V^*(s)$	90	100	81	90	100	0
$\pi^*(s)$	R	R	R, U	R, U	U	self

$\Pi^*(s) = \arg\max [r(s,a) + \gamma V^*(\delta(s,a))]$

state1: $\max[0 + 0.9(100), 0 + 0.9(81)] = 90$ — วิ่งขวา

state2: $\max(0 + 0.9(90), 0 + 0.9(90), 100 + 0.9(0)) = 100$ — วิ่งขวา

Action-Value or Quality (Q)-Function

In practice, the agent can only learn by interacting with the environment, i.e., ***on-line learning***:

- Execute a certain action a_t in state s_t
- Observe the reward r_t and succeeding state s_{t+1}.

If we define the action-value or **Q function**:

Immediate Reward

max future discounted cumulative reward

$$Q(s_t, a_t) = \max_{a_{t+1}, a_{t+2}, \ldots} \left\{ r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \cdots \right\} \tag{4}$$

Then, the optimal policy and value can be determined from

$$\pi^*(s_t) = \arg\max_{a_t} Q(s_t, a_t)$$

$$\pi^*(s) = \arg\max_{a} Q(s, a) \tag{5a}$$

$$V^*(s) = \max_{a} Q(s, a) \tag{5b}$$

One must express $Q(s_t, a_t)$ in the recurrence relation form.

The agent can iteratively approximate the Q function based on ***observed rewards and next states*** of each action taken without the knowledge of $r(s, a)$ and $\delta(s, a)$.

$$Q(s_t, a_t) = \max_{a_{t+1}, a_{t+2}, \ldots} \left\{ r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \cdots \right\}$$

↓ จัดรูป

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \tag{6}$$

↑ observed ↑ observed

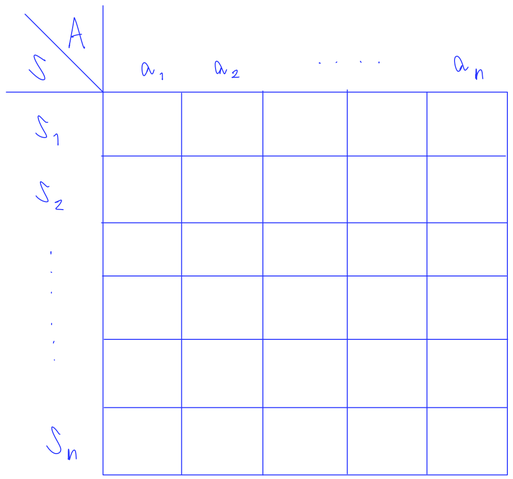
Q-Learning Algorithm

From (6), by observing reward $r = r(s, a)$ and next state $s' = \delta(s, a)$, the learner can iteratively approximate $Q(s, a)$ with:

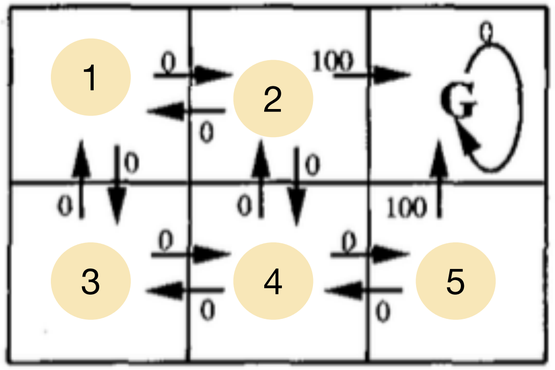
$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a') \qquad (7)$$

$$s \leftarrow s'$$

The optimal policy is determined from (5a).



Q-table at the start of episode 1



	Left	Right	Up	Down
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
G	0	0	0	0

Episode 1 (Initial state 4) เลือก action ที่ให้ค่า Q สูงสุด หรือ random otherwise

t=0, s = 4, a = right, s' = 5

	Left	Right	Up	Down
1				
2				
3				
4		0		
5				
G				

$Q(4,2) \leftarrow 0 + 0.9(0) = 0$

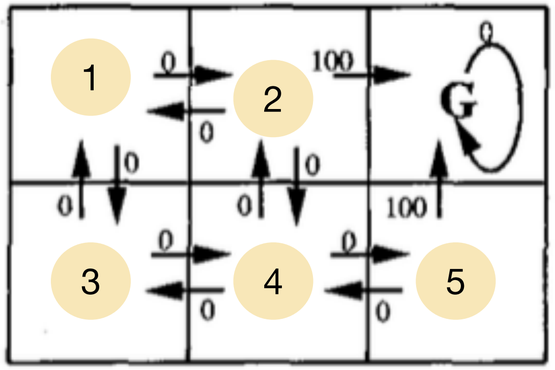
$Q(s,up) = 100 + 0.9(0) = 100$

t=1, s = 5, a = up, s' = G

	Left	Right	Up	Down
1				
2				
3				
4		0		
5			100	
G				

$Q(s, up) \leftarrow 100 + 0.9(0)$

Q-table at the start of episode 2



	Left	Right	Up	Down
1				
2				
3				
4		0		
5			100	
G				

Episode 2 (Initial state 4)

$Q(4,R) = 0 + 0.9 \times 100$

t=0: s = 4, a = right, s' = 5

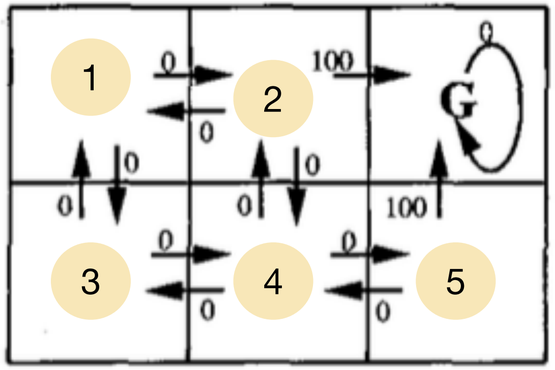
	Left	Right	Up	Down
1				
2				
3				
4		90		
5			100	
G				

t=1: s = 5, a = up, s' = G

	Left	Right	Up	Down
1				
2				
3				
4		90		
5			100	
G				

$Q(s, up) \leftarrow 100 + 0.9 \times 0$

Q-table at the start of episode 3



	Left	Right	Up	Down
1				
2				
3				
4		90		
5			100	
G				

Episode 3 (Initial state 2)

t=0: s = 2, a = down, s' = 4

	Left	Right	Up	Down
1				
2				81
3				
4		90		
5			100	
G				

$Q(2, D) = 0 + 0.9(90)$

$= 81$

t=1: s = 4, a = right, s' = 5

	Left	Right	Up	Down
1				
2				81
3				
4		90		
5			100	
G				

$Q(4, R) = 0 + 0.9(100) = 90$

t=2: s = 5, a = up, s' = G

	Left	Right	Up	Down
1				
2				81
3				
4		90		
5			100	
G				

$Q(5, up) = 100 + 0.9(0) = 100$

Behavior Policy - Experimentation Strategies

The agent tries to learn the ***optimal/target*** policy from its history of interaction with the environment by using the ***behavior policy***.

$$(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \ \ldots \ , (s_t, a_t, r_t), \ \ldots$$

Behavior policy has the following two important strageties:

Exploitation Choose action that maximizes $\hat{Q}$.

Exploration Try different actions other than the current best, which could improve $\hat{Q}$ in a long run.

Examples of Behavior Policy

Greedy action selection policy

Random action selection policy

$\epsilon-$greedy selection policy $\longrightarrow$ Greed with prob. $1-\varepsilon$

Gibbs or ***Boltzmann*** action selection policy

$$\mathbb{P}\{a_i \mid s\} = \frac{e^{\hat{Q}(s,a_i)/\tau}}{\sum_j e^{\hat{Q}(s,a_j)/\tau}}, \quad \tau > 0$$

Math Framework of Reinforcement Learning - Markov Decision Process (MDP)

In general, the reward and state transition functions can have ***non-deterministic*** outcomes.

RL is mathematically formalized under the framework of ***Markov Decision Process (MDP)***.

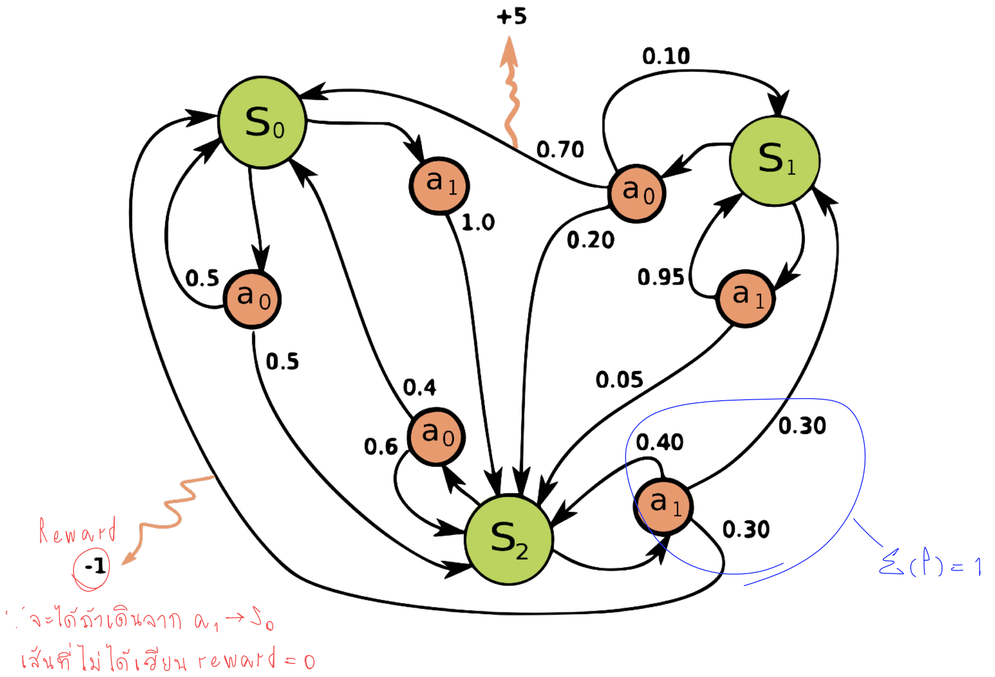
Four elements of MDP:

- State space S
- Action space A
- State-Action transition probabilities
- State-Action reward probabilities

$$\mathbb{P}\{s' \mid s, a\} \triangleq \mathbb{P}\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

$$\mathbb{E}\{r_t(s_t, a_t)\} = \sum_r r\, \mathbb{P}\{r_t = r \mid s_t = s, a_t = a\}$$

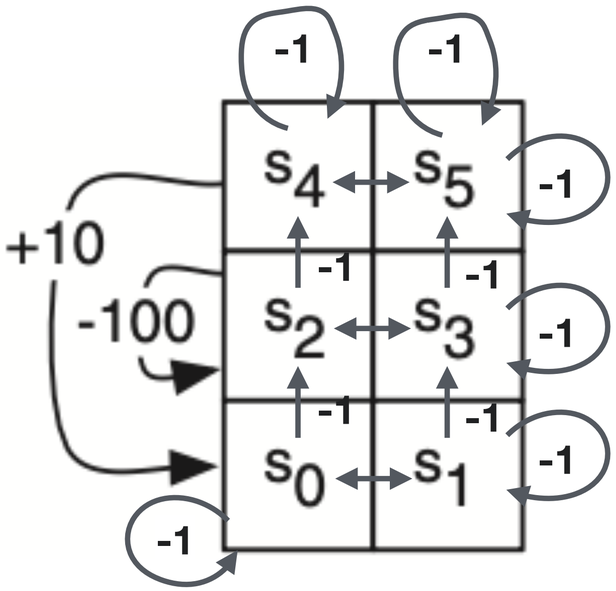
Globally optimal policy is analytically solvable under the perfect knowledge of reward and transition structures.



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows)

https://en.wikipedia.org/wiki/Markov_decision_process

Example



Set of actions and rewards

upC (for "up carefully") The agent goes up, except in states s_4 and s_5, where the agent stays still, and has a reward of -1.

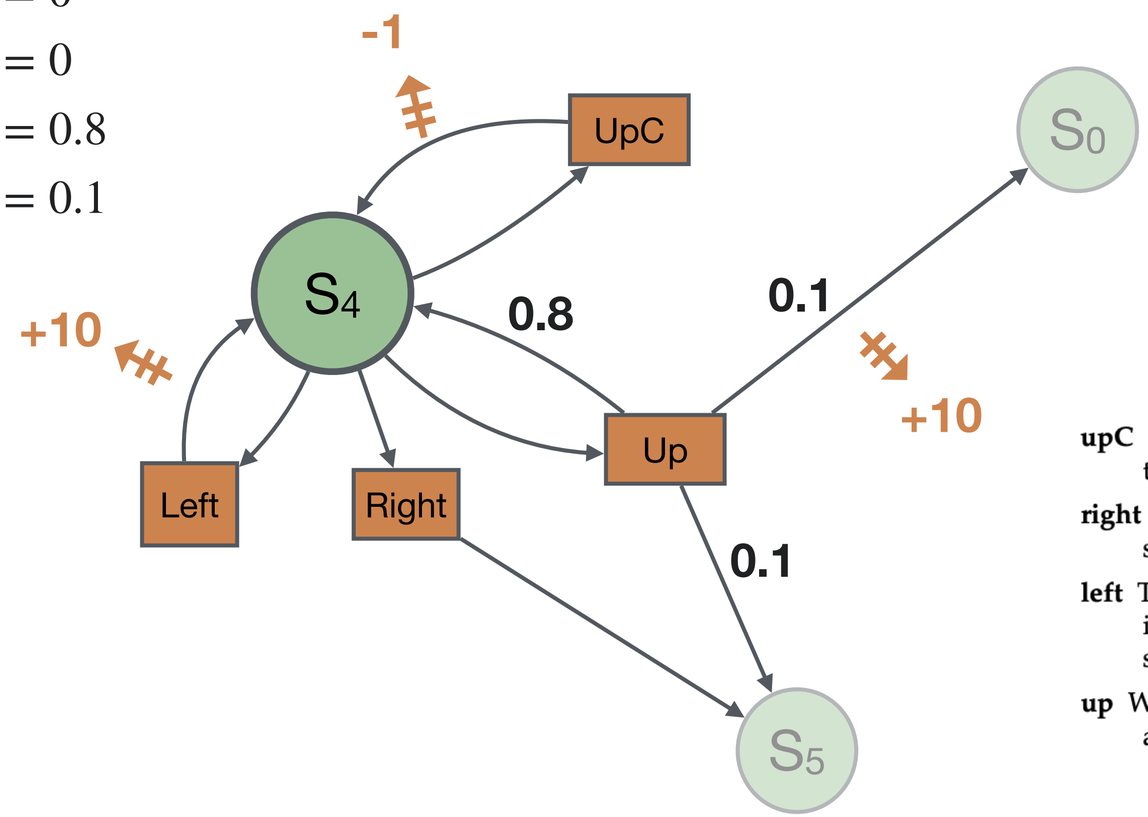
right The agent moves to the right in states s_0, s_2, s_4 with a reward of 0 and stays still in the other states, with a reward of -1.

left The agent moves one state to the left in states s_1, s_3, s_5. In state s_0, it stays in state s_0 and has a reward of -1. In state s_2, it has a reward of -100 and stays in state s_2. In state s_4, it gets a reward of 10 and moves to state s_0.

up With a probability of 0.8 it acts like *upC*, except the reward is 0. With probability 0.1 it acts as a *left*, and with probability 0.1 it acts as *right*.

✳ Determine the policy to maximize the long-term cumulative reward.

$$P(s_0 \mid s_4, \text{Up}) = 0.1$$
$$P(s_1 \mid s_4, \text{Up}) = 0$$
$$P(s_2 \mid s_4, \text{Up}) = 0$$
$$P(s_3 \mid s_4, \text{Up}) = 0$$
$$P(s_4 \mid s_4, \text{Up}) = 0.8$$
$$P(s_5 \mid s_4, \text{Up}) = 0.1$$



upC (for "up carefully") The agent goes up, except in states s_4 and s_5, where the agent stays still, and has a reward of -1.

right The agent moves to the right in states s_0, s_2, s_4 with a reward of 0 and stays still in the other states, with a reward of -1.

left The agent moves one state to the left in states s_1, s_3, s_5. In state s_0, it stays in state s_0 and has a reward of -1. In state s_2, it has a reward of -100 and stays in state s_2. In state s_4, it gets a reward of 10 and moves to state s_0.

up With a probability of 0.8 it acts like *upC*, except the reward is 0. With probability 0.1 it acts as a *left*, and with probability 0.1 it acts as *right*.

Temporal Difference (TD) Error

Consider sequence of observed numeric values $v_1, v_2, v_3, \ldots$

Let A_k be an estimate of the expected value of (most recent) k data points. Then,

$$A_k = \frac{v_1 + v_2 + \cdots + v_k}{k}$$

$$= A_{k-1} + \alpha_k (v_k - A_{k-1}), \quad \alpha_k = \frac{1}{k}$$

most recent data point

previous estimate

$\frac{v_1 + v_2 + \ldots + v_{k-1}}{k-1}$

Temporal difference (TD) error

$v_k > A_{k-1}$

$v_k < A_{k-1}$

- How does A_k change with the latest observed value?
- How much A_k change with the latest observed value?

In RL, more recent values are more accurate and could be weighted more as the learning progress.

Q-Learning in a Non-Deterministic Environment

One form of ***temporal difference learning*** where the TD error is the change in Q value.

The agent learns from the history of interaction as a sequence of ***experiences*** $\langle s, a, r, s' \rangle$ using

current

TD Error

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha[r(s,a) + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)], \quad 0 < \alpha < 1$$

new estimate

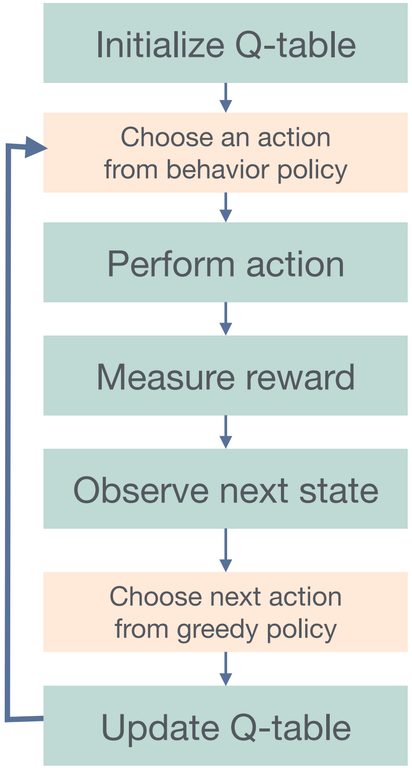
Actual return = Observed reward + Discounted future reward

Current estimated return

- Current action a in state s chosen by using the behavior policy.
- Next state action a' chosen to maximize the actual return (the greedy policy).

Regarded as an ***off-policy*** learning algorithm - Learn an optimal policy regardless of the behavior policy used during learning provided that it explores enough state-action pairs.

```
1:  controller Q-learning(S, A, γ, α)
2:      Inputs
3:          S is a set of states
4:          A is a set of actions
5:          γ the discount
6:          α is the step size
7:      Local
8:          real array Q[S, A]
9:          previous state s
10:         previous action a
11:     initialize Q[S, A] arbitrarily
12:     observe current state s
13:     repeat
14:         select and carry out an action a
15:         observe reward r and state s'
16:         Q[s, a] ← Q[s, a] + α (r + γ max_a' Q[s', a'] − Q[s, a])
17:         s ← s'
18:     until termination
```



Q-learning

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

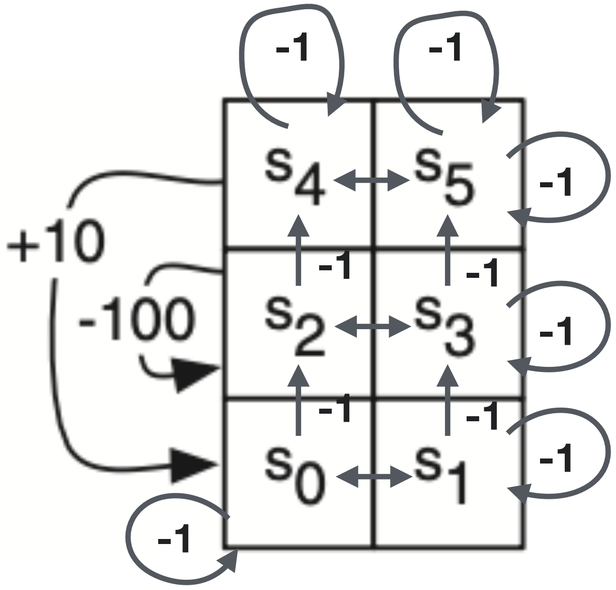
$$\alpha = 0.2,\ \gamma = 0.9$$

upC (for "up carefully") The agent goes up, except in states s_4 and s_5, where the agent stays still, and has a reward of -1.

right The agent moves to the right in states s_0, s_2, s_4 with a reward of 0 and stays still in the other states, with a reward of -1.

left The agent moves one state to the left in states s_1, s_3, s_5. In state s_0, it stays in state s_0 and has a reward of -1. In state s_2, it has a reward of -100 and stays in state s_2. In state s_4, it gets a reward of 10 and moves to state s_0.

up With a probability of 0.8 it acts like *upC*, except the reward is 0. With probability 0.1 it acts as a *left*, and with probability 0.1 it acts as *right*.



Initial Q-table

	UpC	Right	Left	Up
S0	0	0	0	0
S1	0	0	0	0
S2	0	0	0	0
S3	0	0	0	0
S4	0	0	0	0
S5	0	0	0	0

History of interaction

observed

	s	a	r	s'
1	s_0	*upC*	-1	s_2
2	s_2	*up*	0	s_4
3	s_4	*left*	10	s_0
4	s_0	*upC*	-1	s_2

step size

Q-Learning

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

$$\alpha = 0.2,\ \gamma = 0.9$$

History of interaction

	s	a	r	s'
1	s_0	*upC*	-1	s_2
2	s_2	*up*	0	s_4
3	s_4	*left*	10	s_0
4	s_0	*upC*	-1	s_2

Initial Q-table

	UpC	Right	Left	Up
S0	0	0	0	0
S1	0	0	0	0
S2	0	0	0	0
S3	0	0	0	0
S4	0	0	0	0
S5	0	0	0	0

1st step

$\hat{Q}(S_0, UpC) \leftarrow 0 + 0.2(-1 + 0.9(0) - 0) = -0.2$

	UpC	Right	Left	Up
S0	-0.2			
S1				
S2				
S3				
S4				
S5				

2nd step

$\hat{Q}(S_2, up) = 0 + 0.2(0 + 0.9(0) - 0) = 0$

	UpC	Right	Left	Up
S0				
S1				
S2				0
S3				
S4				
S5				

3rd step

	UpC	Right	Left	Up
S0	-0.2			
S1				
S2				
S3				
S4			2	
S5				

4th step

	UpC	Right	Left	Up
S0	-0.36			
S1				
S2				0
S3				
S4			2	
S5				

$\hat{Q}(S_0, UpC) = -0.2 + 0.2(-1 + 0.9(0) - (-0.2))$

$= -0.36$

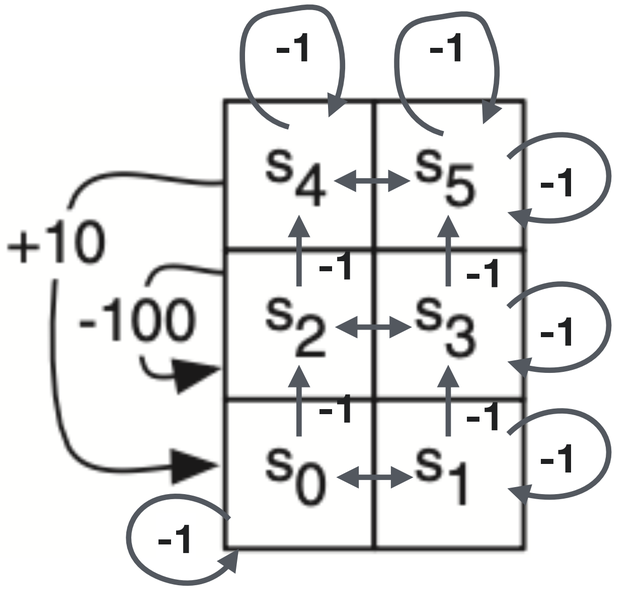
upC (for "up carefully") The agent goes up, except in states s_4 and s_5, where the agent stays still, and has a reward of -1.

right The agent moves to the right in states s_0, s_2, s_4 with a reward of 0 and stays still in the other states, with a reward of -1.

left The agent moves one state to the left in states s_1, s_3, s_5. In state s_0, it stays in state s_0 and has a reward of -1. In state s_2, it has a reward of -100 and stays in state s_2. In state s_4, it gets a reward of 10 and moves to state s_0.

up With a probability of 0.8 it acts like *upC*, except the reward is 0. With probability 0.1 it acts as a *left*, and with probability 0.1 it acts as *right*.

	UpC	Right	Left	Up
S0				x
S1				x
S2	x			
S3				x
S4			x	
S5			x	



Optimal from TD learning $\pi^*(s) = \arg\max_a Q(s,a)$

$\pi^*(s_0) =$ up

$\pi^*(s_1) =$ up

$\pi^*(s_2) =$ upC

$\pi^*(s_3) =$ up

$\pi^*(s_4) =$ left

$\pi^*(s_5) =$ left

Generalizing from Examples - Deep Q Learning

Q-learning requires that all state-action pairs are to be visited infinitely often for $\hat{Q}$ to converge to actual Q.

Impossible to learn when the action-state space gets large (infinite or prohibitively large), the cost of execution actions is high.

Substitute a (deep) neural network for the lookup table and use each $\hat{Q}(s, a)$ update as a training example.

- One network with state and action as input.
- Train a separate network for each action with states as input.