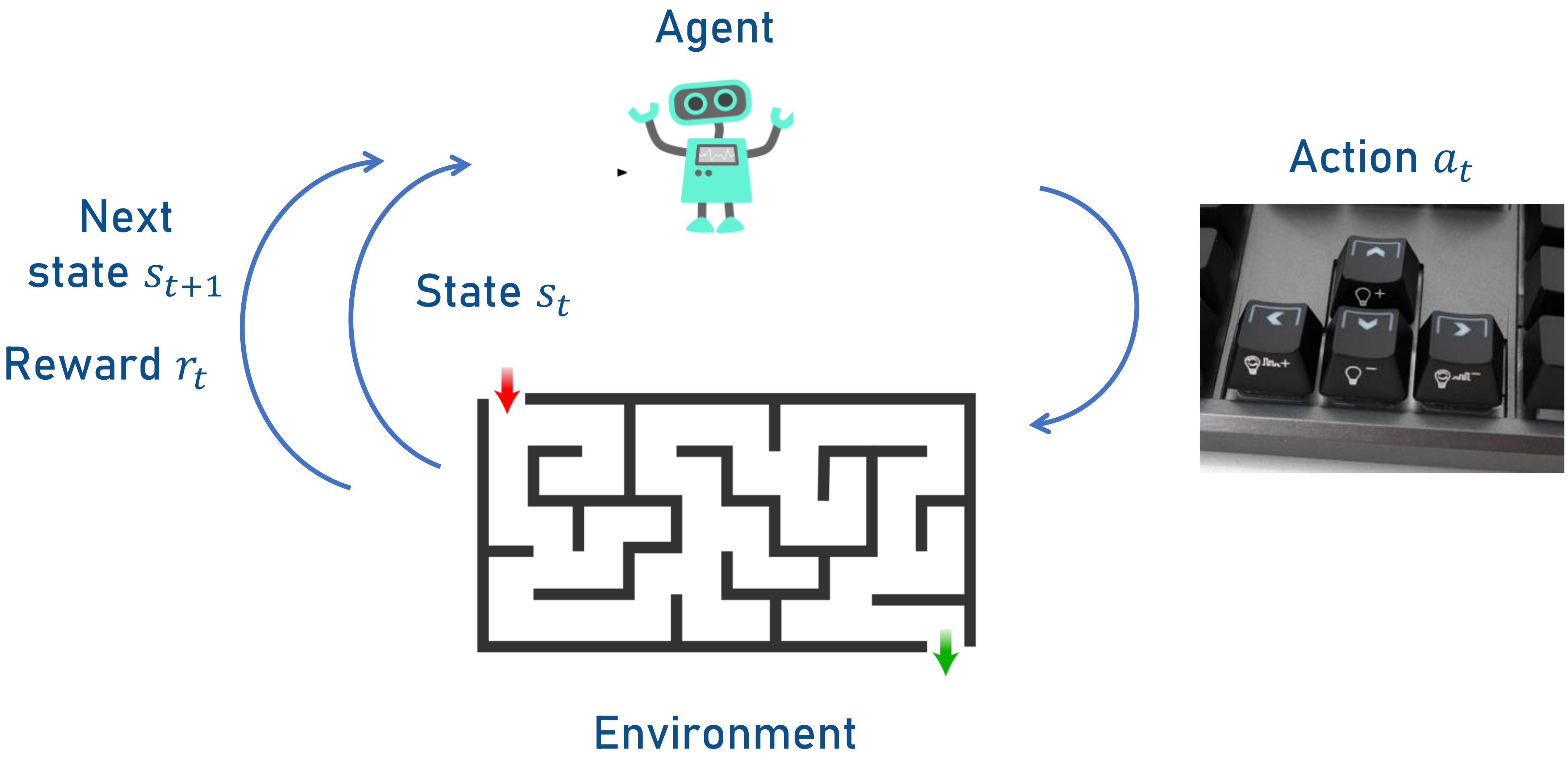
Reinforcement Learning 2

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING
인공지능학과
Department of
Artificial Intelligence

정 우 환 (whjung@hanyang.ac.kr)

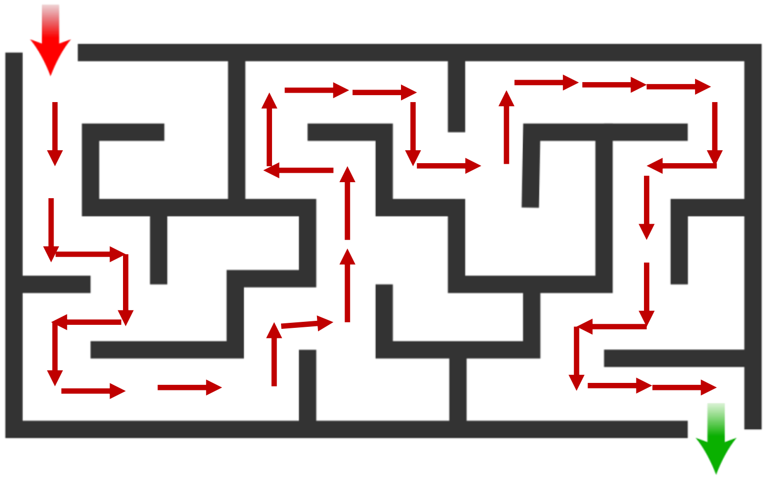
Fall 2022

Reinforcement Learning

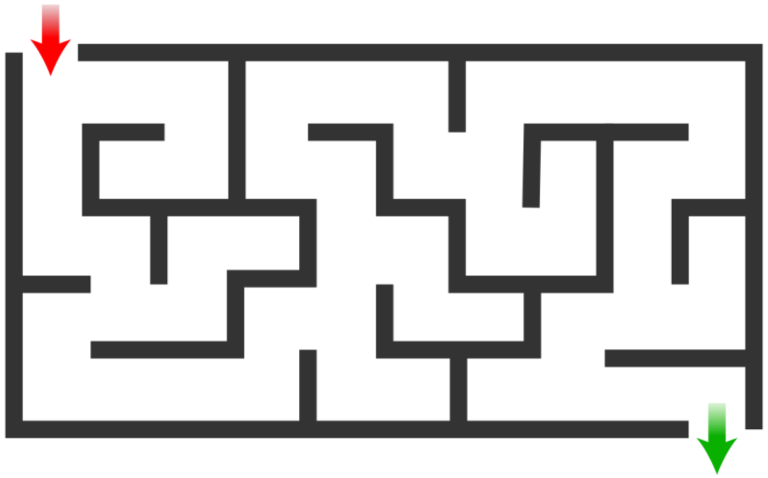


Sparse signal

Labels in supervised learning



Rewards in reinforcement learning



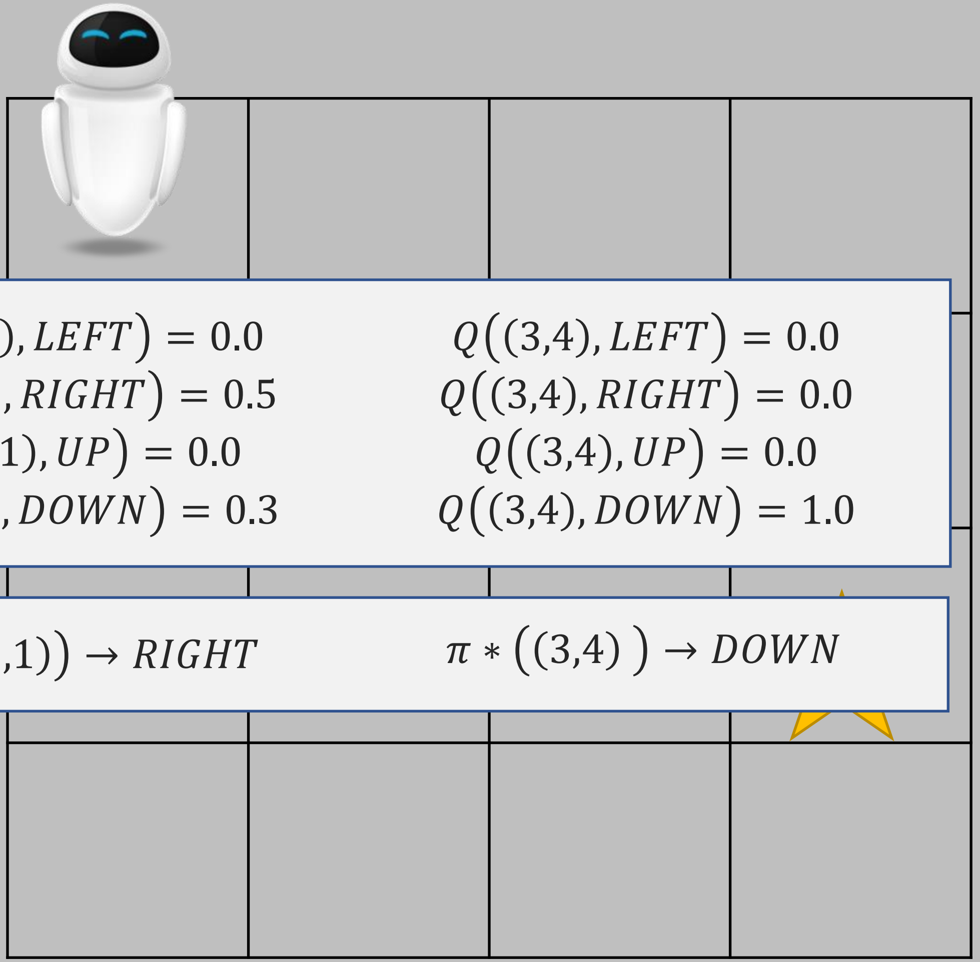
$-T$

Q-learning

Value functions

- Total reward $R_t = r_{t+1} + r_{t+2} + \cdots$
- Value functions measure the expected total reward after t
  - Value of a **state** $V(s)$
    - $V(s) = E[r_{t+1} + r_{t+2} + \cdots | s_t = s]$
  - Value of **of taking an action in a state** $Q(s, a)$
    - $Q(s, a) = E[r_{t+1} + r_{t+2} + \cdots | a_t = a, s_t = s]$
- Policy π: mapping from **state** to **action**
  - Optimal policy $\pi^*(s) = \arg\max_a Q(s, a)$

Q-Learning



$Q\big((1{,}1), LEFT\big) = 0.0$
$Q\big((1{,}1), RIGHT\big) = 0.5$
$Q\big((1{,}1), UP\big) = 0.0$
$Q\big((1{,}1), DOWN\big) = 0.3$

$Q\big((3{,}4), LEFT\big) = 0.0$
$Q\big((3{,}4), RIGHT\big) = 0.0$
$Q\big((3{,}4), UP\big) = 0.0$
$Q\big((3{,}4), DOWN\big) = 1.0$

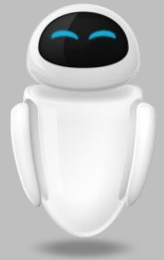
$\pi * \big((1{,}1)\big) \rightarrow RIGHT$

$\pi * \big((3{,}4)\big) \rightarrow DOWN$

Q-Function (State-action value) Q(state,action)

State (x,y)

Action



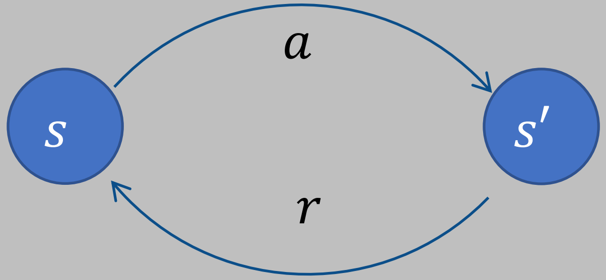
Future reward

$$Q(s_t, a_t) = E[r_{t+1} + r_{t+2} + \cdots | a_t, s_t]$$

Optimal policy

$$\pi^*(s) = \arg\max_a Q(s, a)$$

Recurrence equation



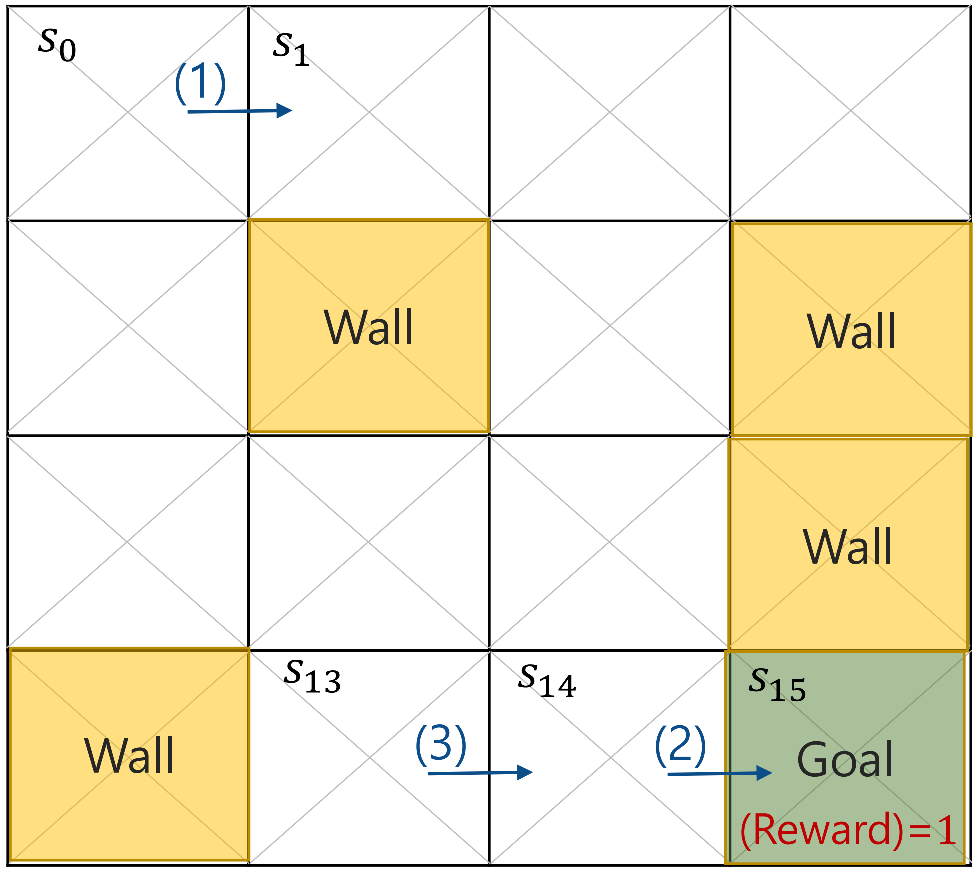
$$Q(s, a) = r + \max_{a'} Q(s', a')$$

Joongheon Kim (https://youtu.be/m1FC3dMmY78)

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

Q-Learning

16 states and 4 actions (U, D, L, R)



- Initial status
  - $Q(s,a) = 0$ for all s, a
  - Reward are all zero except in s_{15}

Case (1) →

$$Q(s_0, R) = r + \max_{a'} Q(s_1, a') = 0 + \max_{a'}\{0,0,0,0\} = 0$$

Case (2) →

$$Q(s_{14}, R) = 1 + \max_{a'} Q(s_{15}, a') = 0 + \max_{a'}\{0,0,0,0\} = 1$$

Case (3) →

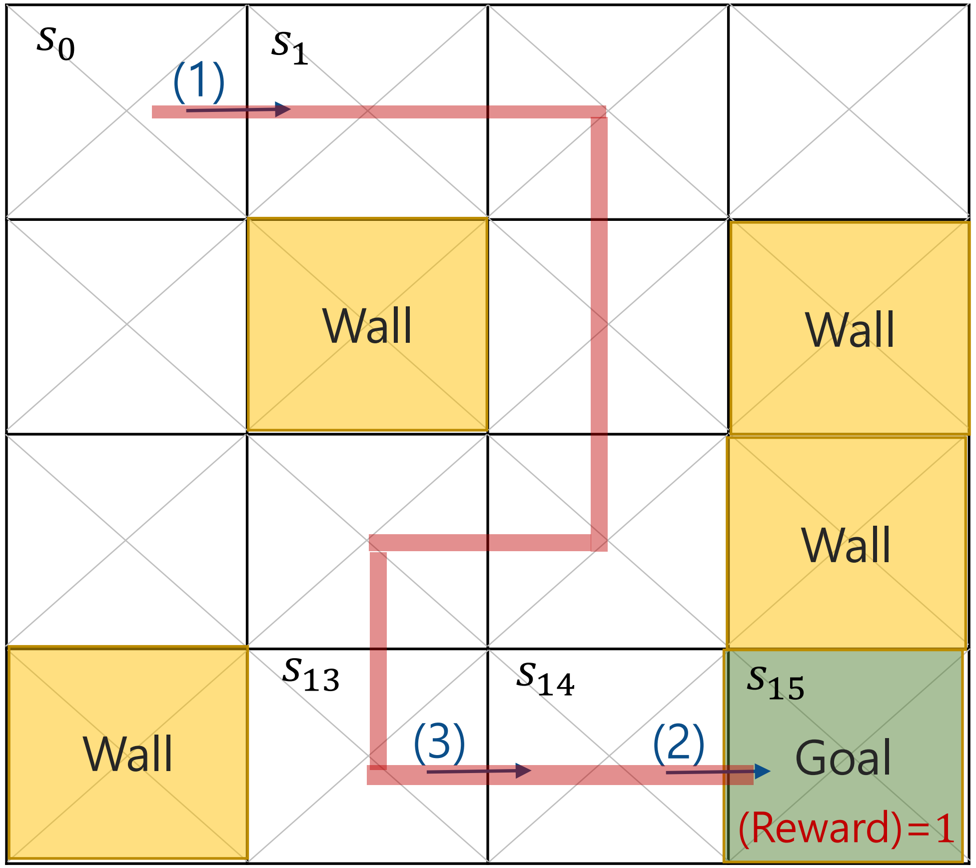
$$Q(s_{13}, R) = r + \max_{a'} Q(s_{14}, a') = 0 + \max_{a'}\{0,0,1,0\} = 1$$

Joongheon Kim (https://youtu.be/m1FC3dMmY78)

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

Q-Learning

16 states and 4 actions (U, D, L, R)



- Initial status
  - $Q(s,a) = 0$ for all s, a
  - Reward are all zero except in s_{15}

Case (1) $\rightarrow$

$$Q(s_0, R) = r + \max_{a'} Q(s_1, a') = 0 + \max_{a'}\{0,0,0,0\} = 0$$

Case (2) $\rightarrow$

$$Q(s_{14}, R) = 1 + \max_{a'} Q(s_{15}, a') = 0 + \max_{a'}\{0,0,0,0\} = 1$$

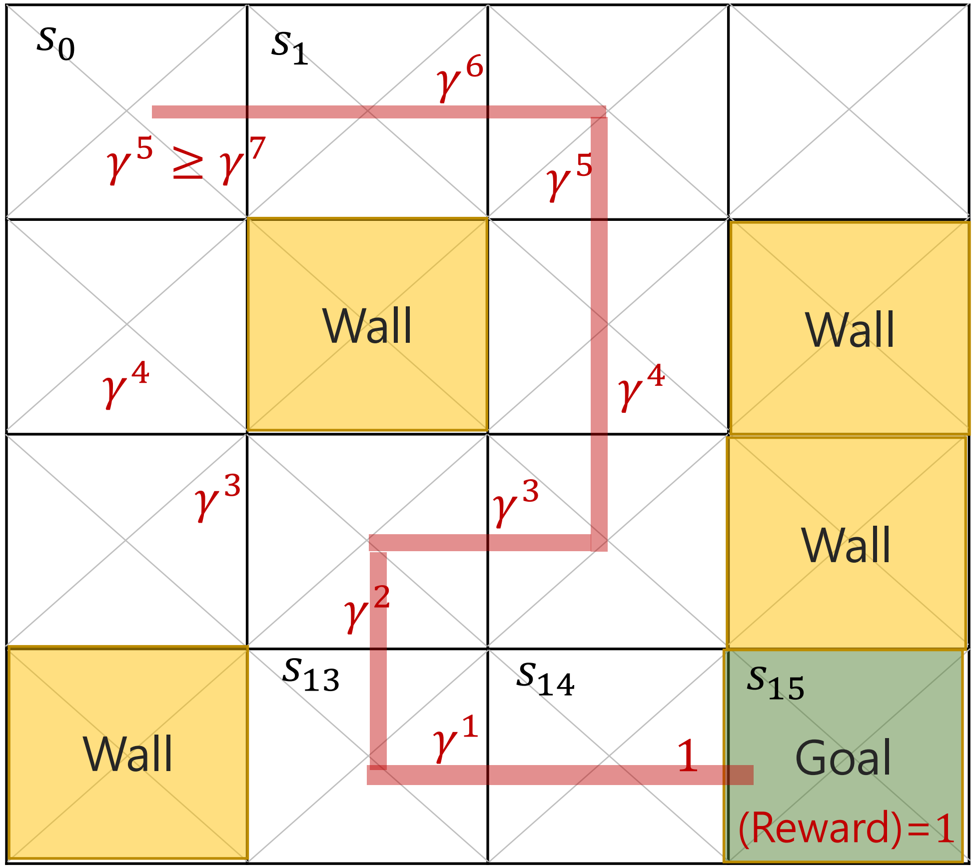
Case (3) $\rightarrow$

$$Q(s_{13}, R) = r + \max_{a'} Q(s_{14}, a') = 0 + \max_{a'}\{0,0,1,0\} = 1$$

Joongheon Kim (https://youtu.be/m1FC3dMmY78)

Q-Learning: Discounted reward

16 states and 4 actions (U, D, L, R)



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

Joongheon Kim (https://youtu.be/m1FC3dMmY78)

Update with discount factor γ
$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$

Q-Learning: Temporal Difference

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \Bigg)}^{\text{temporal difference}}$$

Range of α: $0 < \alpha < 1$

If learning rate $\alpha = 1$, $Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$
If learning rate $\alpha = 0$, $Q(s,a) \leftarrow Q(s,a)$

Q-learning

- For each s, a, initialize table entry $Q(s,a) \leftarrow 0$
- Do until Q converges
  - Initialize s
  - Do until s *is terminal*
    - Select an action a using policy π derived from Q
    - Take action a
    - Receive immediate reward r
    - Observe the new state s'
    - $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right]$
    - $s \leftarrow s'$

Iterative update
Discounted reward
Temporal difference

Policy π

- Testing phase
  - Optimal policy $\pi^*(s) = \arg\max_a Q(s, a)$
- Training phase (**ϵ-greedy**)
  - **Exploration-Exploitation**
  - Exploration allows an agent to improve its current knowledge
  - Exploitation chooses the greedy action to get the most reward by exploiting the agent's current action-value estimates

$$\text{Action at time(t)} \begin{cases} \max Q_t(a) & \text{with probability } 1-\epsilon \\ \text{any action } (a) & \text{with probability } \epsilon \end{cases}$$

https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/

Q-learning

- For each s, a, initialize table entry $Q(s,a) \leftarrow 0$
- Do until Q converges
  - Initialize s
  - Do until s is terminal
    - Draw a random value $v \sim Uniform(0,1)$
    - If $v < \varepsilon$
      - Randomly select a
    - Else:
      - $a = \underset{a'}{\operatorname{argmax}}\, Q(s, a')$
    - Take action a
    - Receive immediate reward r
    - Observe the new state s'
    - $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s,a)\right]$
    - $s \leftarrow s'$

ϵ-greedy

Implementing a Q-learning algorithm in Python

Taxi-v3

https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

Environments Documentation

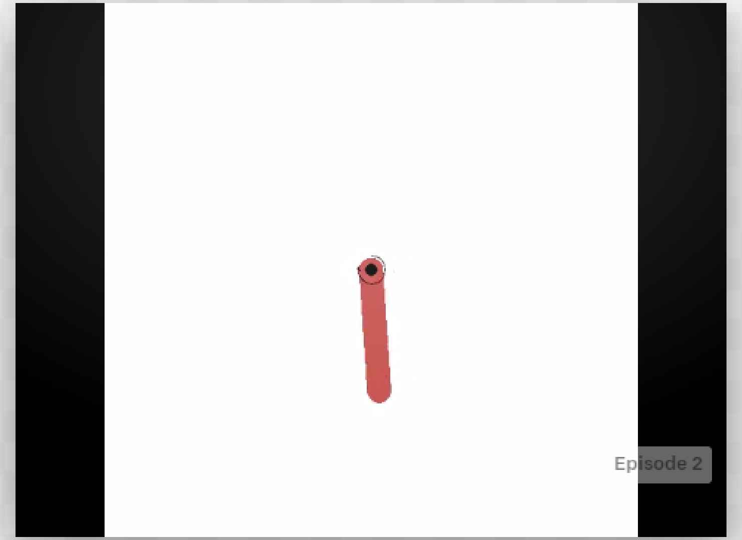
RandomAgent on CartPole-v1

Gym

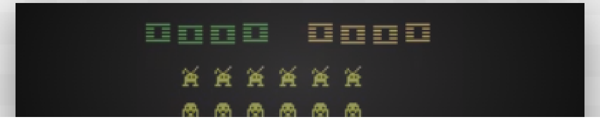
Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

View documentation ›

View on GitHub ›



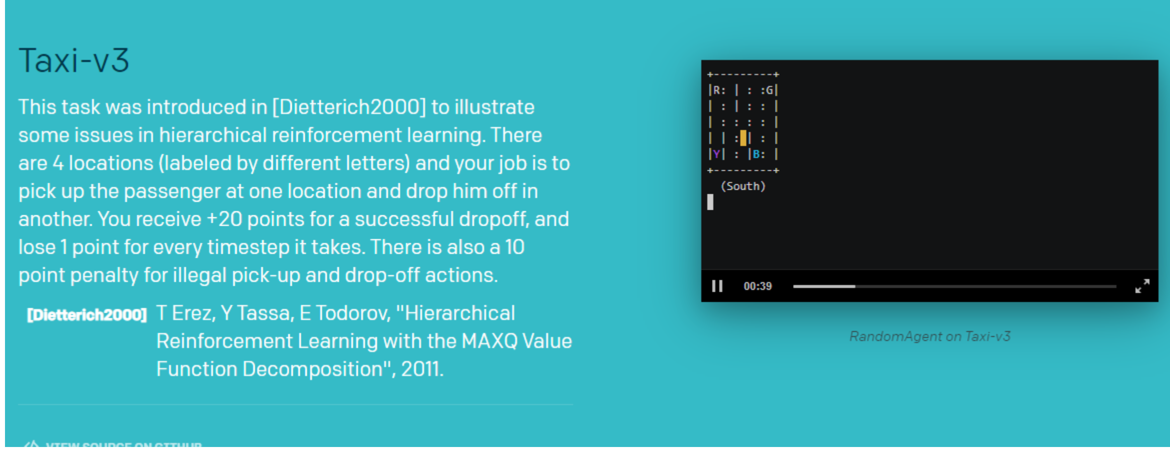
RandomAgent on Pendulum-v0



Open source interface to reinforcement learning tasks.

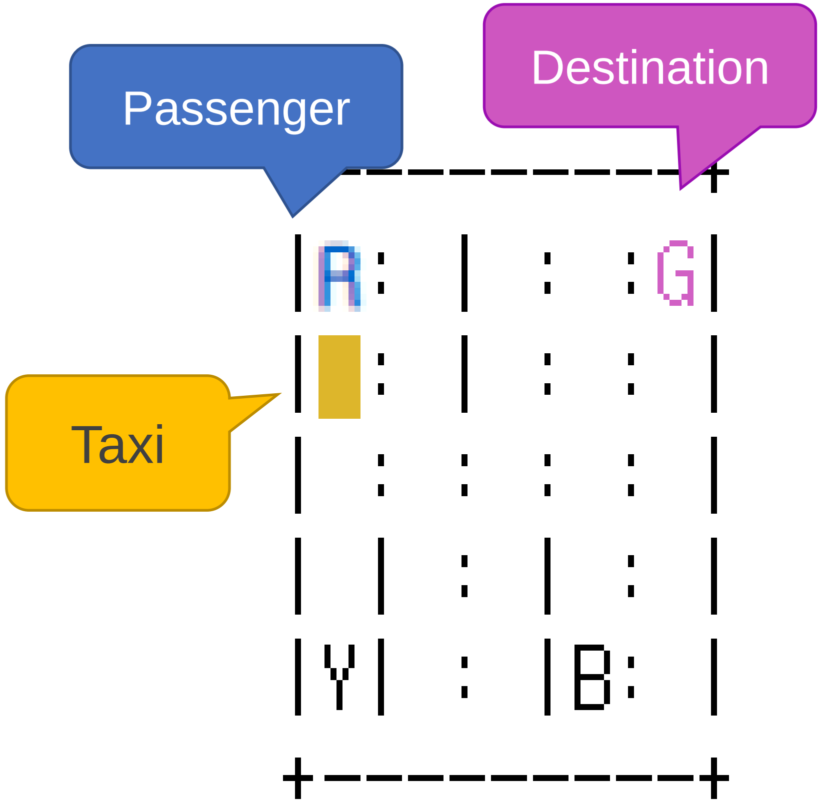
The gym library provides an easy-to-use suite of reinforcement

Taxi-v3


Taxi-v3

This task was introduced in [Dietterich2000] to illustrate some issues in hierarchical reinforcement learning. There are 4 locations (labeled by different letters) and your job is to pick up the passenger at one location and drop him off in another. You receive +20 points for a successful dropoff, and lose 1 point for every timestep it takes. There is also a 10 point penalty for illegal pick-up and drop-off actions.

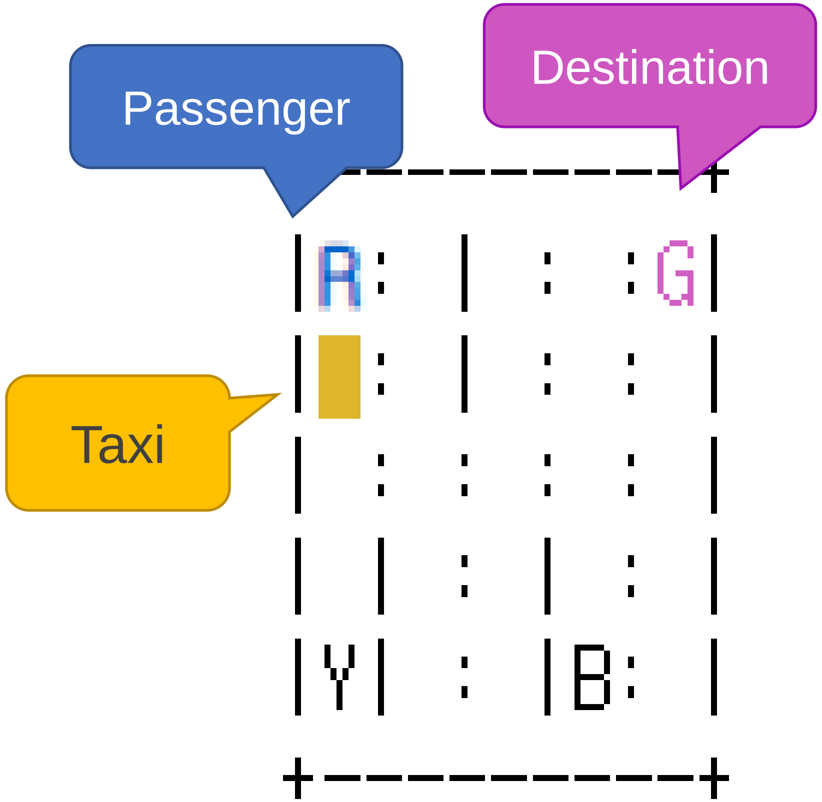
[Dietterich2000] T Erez, Y Tassa, E Todorov, "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", 2011.



https://gym.openai.com/envs/Taxi-v3/
https://github.com/openai/gym/blob/master/gym/envs/toy_text/taxi.py

- 4 designated locations: R(ed), G(reen), Y(ellow), and B(lue)
- When the episode starts
  - The taxi starts off at a random location
  - The passenger is at a random location
- Goal
  - Taxi picks up the passenger and drives to the destination with the minimum operations

Taxi-v3



- 6 actions
  - 0: move south
  - 1: move north
  - 2: move east
  - 3: move west
  - 4: pickup passenger
  - 5: drop off passenger
- States
  - (Taxi row, taxi col, passenger_location, destination)
  - $(5 \times 5 \times (4 + 1) \times 4) = 500$ possible states
    - Passenger location: RGBY + taxi
- Rewards
  - Per step reward: -1
  - Delivering the passenger: +20
  - Illegal pickup and drop-off: -10

Taxi-v3

- Initialize environment

```
In [1]: import gym
```

```
In [2]: env = gym.make("Taxi-v3").env
```

- Render environment

```
In [3]: env.render()

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
In [4]: env.reset()
        env.render()

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
In [5]: env.reset()
        env.render()

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Action space and state space

```
print("Action Space {}".format(env.action_space))
print("State Space {}".format(env.observation_space))
```

```
Action Space Discrete(6)
State Space Discrete(500)
```

States
(Taxi row, taxi col, passenger_location, destination)

```
state = env.encode(3, 1, 2, 0)
print("State:", state)

env.s = state
env.render()
```

```
State: 328
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
state = env.encode(3, 1, 3, 2) #
print("State:", state)

env.s = state
env.render()
```

```
State: 334
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Reward table P

6 actions

0: move south
1: move north
2: move east
3: move west
4: pickup passenger
5: drop off passenger

```
state = env.encode(3, 1, 2, 0)
print("State:", state)

env.s = state
env.render()
```

```
State: 328
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
In [22]: env.P[328]
```

```
Out[22]: {0: [(1.0, 428, -1, False)],
 1: [(1.0, 228, -1, False)],
 2: [(1.0, 348, -1, False)],
 3: [(1.0, 328, -1, False)],
 4: [(1.0, 328, -10, False)],
 5: [(1.0, 328, -10, False)]}
```

Brute force algorithm

- Brute-force search is a very general problem-solving technique
  - Enumerates all possible candidates
  - Checks whether each candidate satisfies the problem's statement

```python
env.s = 328  # set environment to illustration's state

epochs = 0
penalties, reward = 0, 0

frames = [] # for animation

done = False

while not done:
    action = env.action_space.sample()
    state, reward, done, info = env.step(action)

    if reward == -10:
        penalties += 1

    # Put each rendered frame into dict for animation
    frames.append({
        'frame': env.render(mode='ansi'),
        'state': state,
        'action': action,
        'reward': reward
        }
    )

    epochs += 1


print("Timesteps taken: {}".format(epochs))
print("Penalties incurred: {}".format(penalties))
```

```
Timesteps taken: 620
Penalties incurred: 208
```

Brute force algorithm

```
from IPython.display import clear_output
from time import sleep

def print_frames(frames, s = .01):
    for i, frame in enumerate(frames):
        clear_output(wait=True)
        print(frame['frame'])
        print(f"Timestep: {i + 1}")
        print(f"State: {frame['state']}")
        print(f"Action: {frame['action']}")
        print(f"Reward: {frame['reward']}")
        sleep(s)
```

```
print_frames(frames)
```

Q-learning

- Q-table size: $(\#states, \#actions)$

```
import numpy as np
q_table = np.zeros([env.observation_space.n, env.action_space.n])
```

Q-learning: Training

- For each s, a, initialize table entry $Q(s,a) \leftarrow 0$
- Do until Q converges
  - Initialize s
  - Do until s is terminal
    - Draw a random value $v \sim Uniform(0,1)$
    - If $v < \varepsilon$
      - Randomly select a
    - Else:
      - $a = \underset{a'}{\text{argmax}}\, Q(s,a')$
    - Take action a
    - Receive immediate reward r
    - Observe the new state s'
    - $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    - $s \leftarrow s'$

```python
from IPython.display import clear_output

# Hyperparameters
alpha = 0.1
gamma = 0.6
epsilon = 0.1

# For plotting metrics
all_epochs = []
all_penalties = []

for i in range(1, 100001):
    state = env.reset()

    epochs, penalties, reward, = 0, 0, 0
    done = False

    while not done:
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample() # Explore action space
        else:
            action = np.argmax(q_table[state]) # Exploit learned values

        next_state, reward, done, info = env.step(action)

        old_value = q_table[state, action]
        next_max = np.max(q_table[next_state])

        new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
        q_table[state, action] = new_value

        if reward == -10:
            penalties += 1

        state = next_state
        epochs += 1

    if i % 100 == 0:
        clear_output(wait=True)
        print(f"Episode: {i}")
```

Q-learning: Test

$$\pi^*(s) = \arg\max_a Q(s,a)$$

```python
total_epochs, total_penalties = 0, 0
episodes = 100

for _ in range(episodes):
    state = env.reset()
    epochs, penalties, reward = 0, 0, 0

    done = False

    while not done:
        action = np.argmax(q_table[state])
        state, reward, done, info = env.step(action)

        if reward == -10:
            penalties += 1

        epochs += 1

    total_penalties += penalties
    total_epochs += epochs

print(f"Results after {episodes} episodes:")
print(f"Average timesteps per episode: {total_epochs / episodes}")
print(f"Average penalties per episode: {total_penalties / episodes}")
```

```
Results after 100 episodes:
Average timesteps per episode: 13.23
Average penalties per episode: 0.0
```

Q-learning:
Test (print result)

```
frames = []

state = env.reset()
epochs, penalties, reward = 0, 0, 0

done = False

while not done:
    action = np.argmax(q_table[state])
    state, reward, done, info = env.step(action)

    if reward == -10:
        penalties += 1

    epochs += 1
    frames.append({
        'frame': env.render(mode='ansi'),
        'state': state,
        'action': action,
        'reward': reward
        }
    )

total_penalties += penalties
total_epochs += epochs
```

```
print_frames(frames, s = 0.1)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

Timestep: 12
State: 85
Action: 5
Reward: 20
```

References

- Fei-Fei Li & Justin Johnson & Serena Yeung, CS231n Lecture 14: Reinforcement Learning, Stanford University
- https://sumniya.tistory.com/ 숨니의 무작정 따라하기
- https://youtu.be/m1FC3dMmY78 Joongheon Kim, Korea University
- https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/