Deep Reinforcement Learning 1: Deep Q-Net

한양대학교 ERICA 소프트웨어융합대학 COLLEGE OF COMPUTING | 인공지능학과 Department of Artificial Intelligence

정 우 환 (whjung@hanyang.ac.kr)

Fall 2022

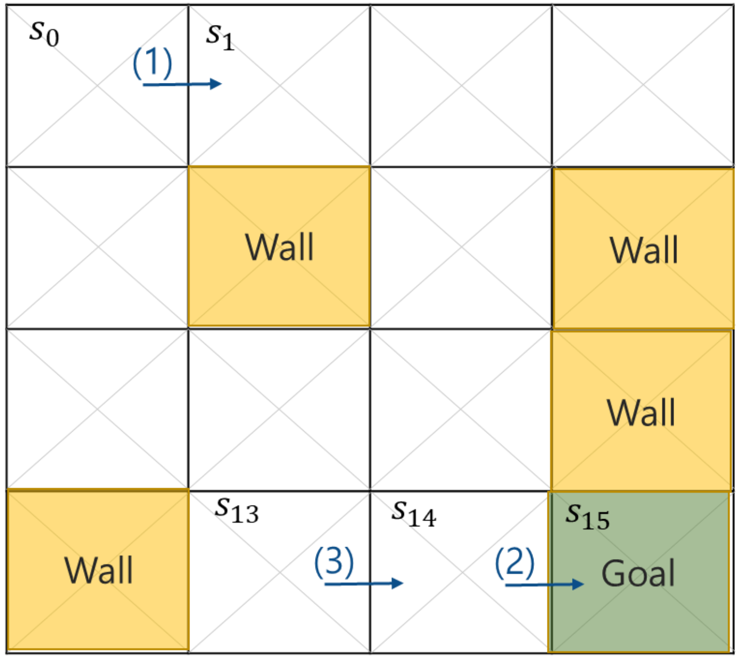
Outline

- Q-learning
- Naïve DQN (Deep Q-Network)
- Deep Q-Network

Q-learning (Recap)

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

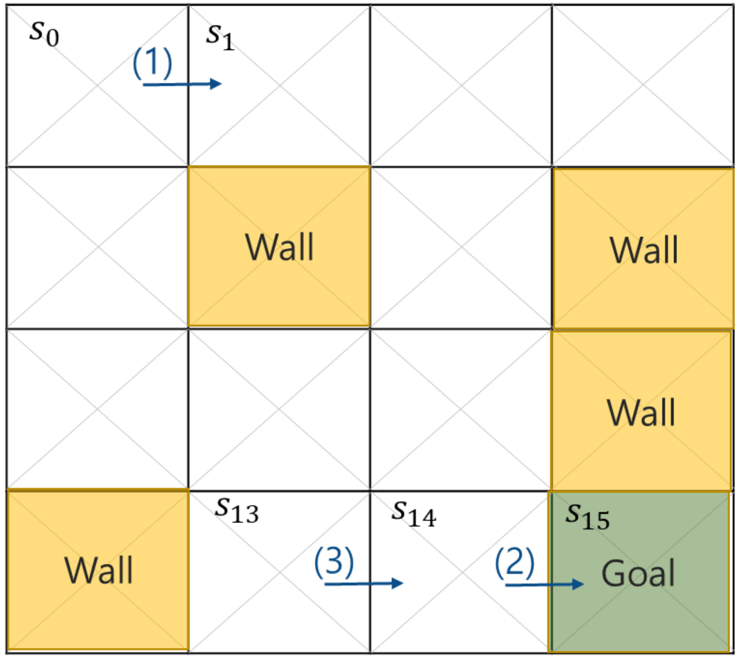
16 states and 4 actions (U, D, L, R)



Q-table

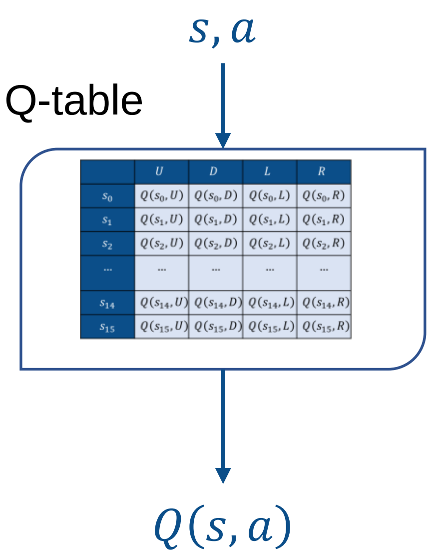
	U	D	L	R
s_0: (1,1)	$Q(s_0, U)$	$Q(s_0, D)$	$Q(s_0, L)$	$Q(s_0, R)$
$s1$: (1,2)	$Q(s_1, U)$	$Q(s_1, D)$	$Q(s_1, L)$	$Q(s_1, R)$
$s2$: (1,3)	$Q(s_2, U)$	$Q(s_2, D)$	$Q(s_2, L)$	$Q(s_2, R)$
…	…	…	…	…
$s14$: (4,3)	$Q(s_{14}, U$	$Q(s_{14}, D)$	$Q(s_{14}, L)$	$Q(s_{14}, R)$
$s15$: (4,4)	$Q(s_{15}, U$	$Q(s_{15}, D)$	$Q(s_{15}, L)$	$Q(s_{15}, R)$

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



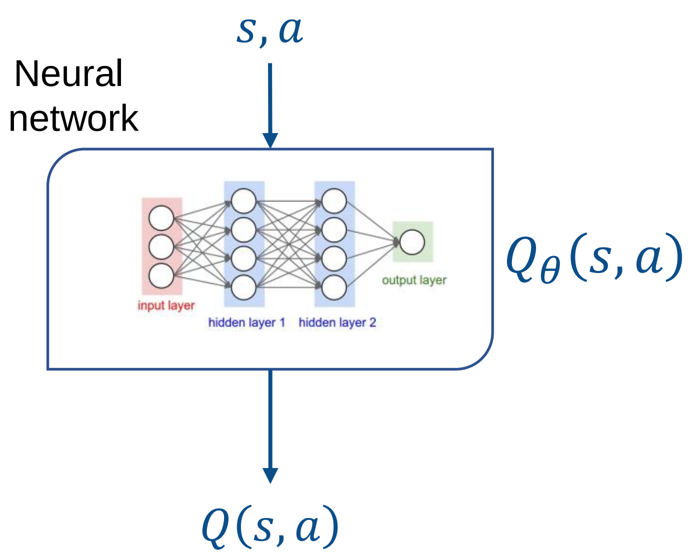
Naïve DQN

Q-learning (Table vs NN)

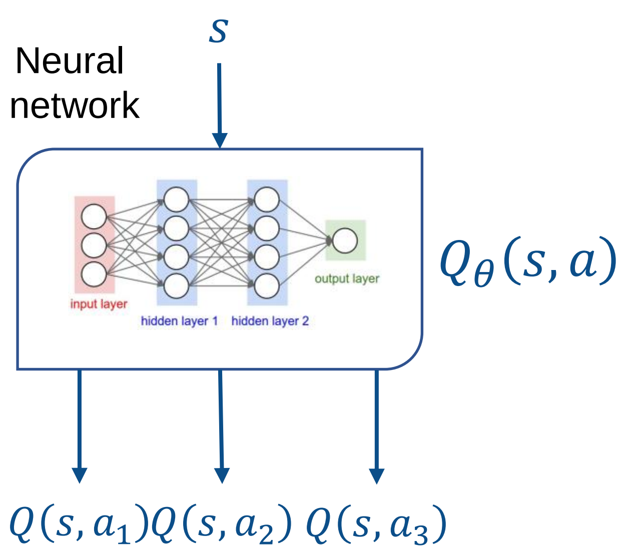


	U	D	L	R
s_0	$Q(s_0, U)$	$Q(s_0, D)$	$Q(s_0, L)$	$Q(s_0, R)$
s_1	$Q(s_1, U)$	$Q(s_1, D)$	$Q(s_1, L)$	$Q(s_1, R)$
s_2	$Q(s_2, U)$	$Q(s_2, D)$	$Q(s_2, L)$	$Q(s_2, R)$
...	...	...	...	...
s_{14}	$Q(s_{14}, U)$	$Q(s_{14}, D)$	$Q(s_{14}, L)$	$Q(s_{14}, R)$
s_{15}	$Q(s_{15}, U)$	$Q(s_{15}, D)$	$Q(s_{15}, L)$	$Q(s_{15}, R)$

Q-learning (Table)



DQN (action-in)



DQN (action-out)

Naïve DQN

Model: $Q_\theta(s_t, a_t)$

Training data: $\langle s_t, a_t, r_t, s_{t+1} \rangle$

Loss function: $\mathcal{L}(\theta) = ||y_t - Q_\theta(s_t, a_t)||_2^2$ where $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$

Naïve DQN

Training data: $\langle s_t, a_t, r_t, s_{t+1} \rangle$

Model: $Q_\theta(s_t, a_t)$

Loss function: $\mathcal{L}(\theta) = ||y_t - Q_\theta(s_t, a_t)||_2^2$

where $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$

- Initialize $Q_\theta(s, a)$
- 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:
      - $a = \operatorname{argmax}_{a'} Q_\theta(s, a)(s, a')$
    - Take action a
    - Receive immediate reward r
    - Observe the new state s'
    - Compute target value $y \leftarrow r + \gamma \max_{a'} Q_\theta(s', a')$
    - SGD to minimize $L(\theta) = ||y - Q_\theta(s, a)||_2^2$
    - $s \leftarrow s'$

Q-Net

```python
class QNet (nn.Module):
    def __init__(self, num_states, num_actions):
        super().__init__()

        self.layers = nn.Sequential(
            nn.Embedding(num_states, 4),
            nn.ReLU(),
            nn.Linear(4, 50),
            nn.ReLU(),
            nn.Linear(50, num_actions)
        )


    def forward(self, x):
        #print(x)
        x = self.layers(x)
        #print(x)
        return x
```

```python
qnet = QNet(env.observation_space.n, env.action_space.n)
optimizer = torch.optim.SGD(qnet.parameters(), lr = 0.01)
criteria = nn.MSELoss()
```

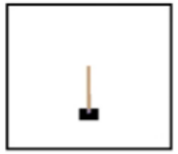
$$\text{minimize } L(\theta) = \|y - Q_\theta(s,a)\|_2^2$$

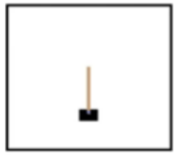
$$y = r + \gamma \max_{a'} Q_\theta(s', a')$$

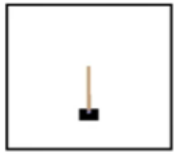
```python
for i in tqdm(range(1, 1001)):
    #epsilon = max(eps_p ** i, 0.1)
    state = env.reset()

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

    step_i = 0
    loss_i = 0
    while not done:
        step_i += 1
        state_t = torch.LongTensor([state])

        if random.uniform(0, 1) < epsilon          :
            action = env.action_space.sample() # Explore action space
        else:
            with torch.no_grad():
                q_hat = qnet(state_t)
                action = torch.argmax(q_hat).item() # Exploit learned values


        next_state, reward, done, info = env.step(action)
        next_state_t = torch.LongTensor([next_state])


        y_t = torch.Tensor([reward])
        if not done:
            with torch.no_grad():
                q_target = qnet(next_state_t)
                y_t = y_t + gamma * q_target.max()


        q_hat = qnet(state_t)
        q_hat = q_hat[:,action]

        optimizer.zero_grad()
        loss = criteria(q_hat, y_t)
        loss.backward()
        optimizer.step()
        loss_i += loss.item()
```

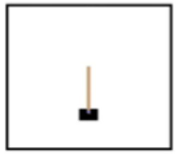
Problems of the Naïve DQN

Problems of the Naïve DGN 1: Correlated samples









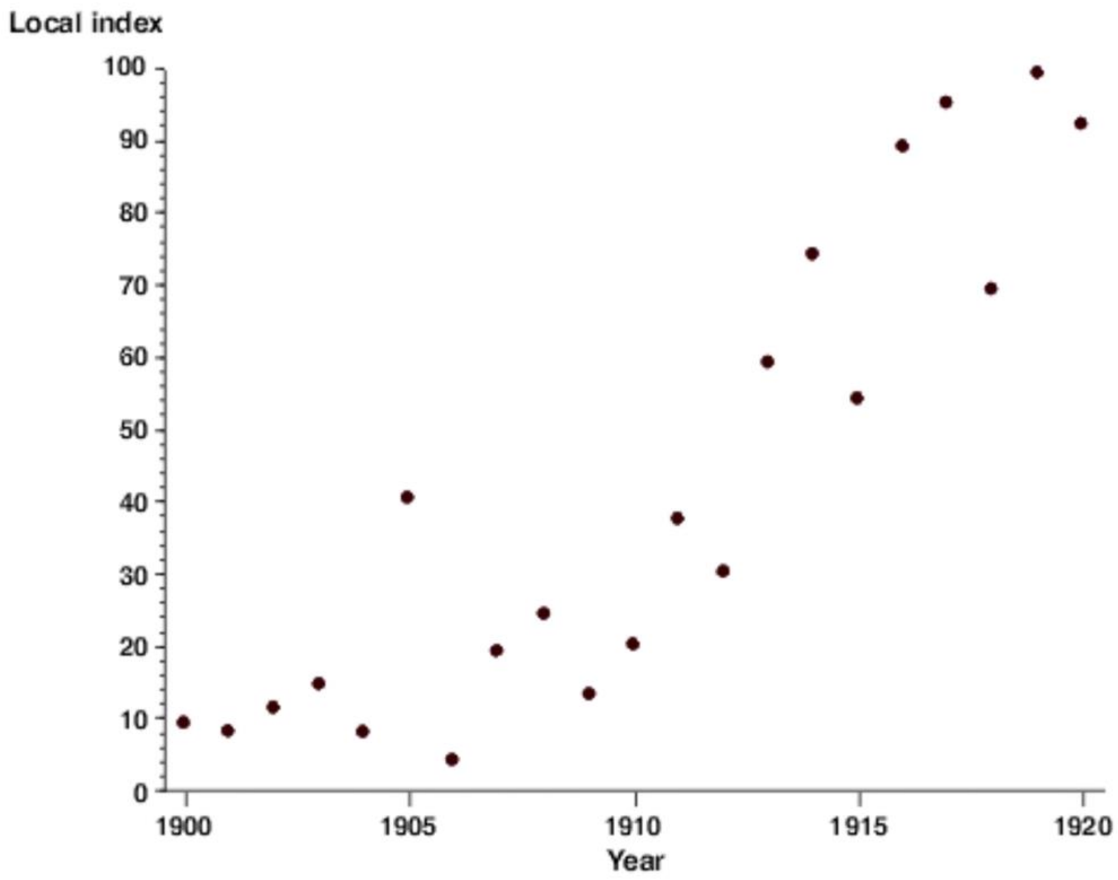
Algorithm 1 Deep Q-learning

Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

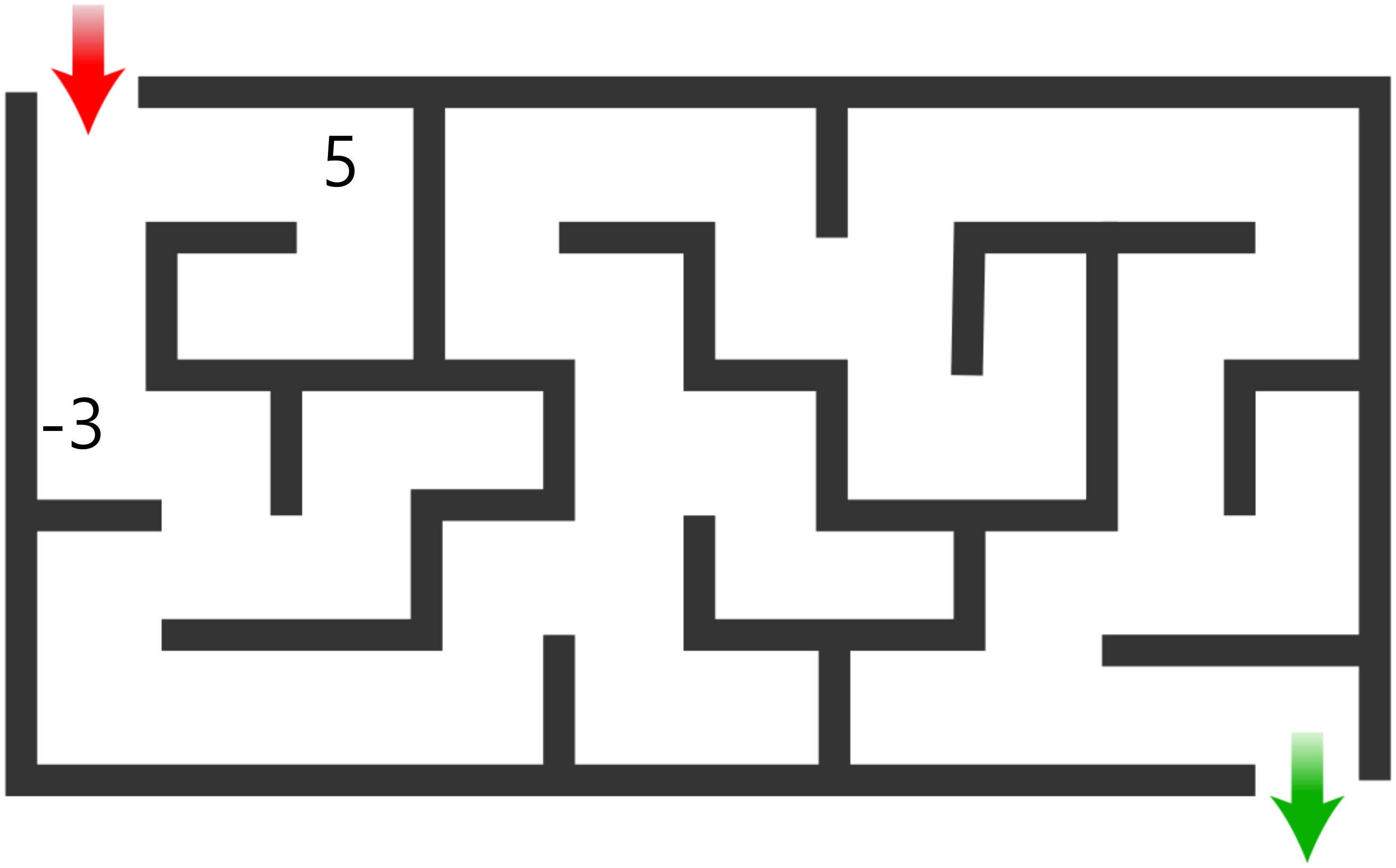
Playing Atari with Deep Reinforcement Learning, Arxiv 2013
김성훈 교수님 유튜브 (https://youtu.be/S1Y9eys2bdg)

Problems of the Naïve DGN 1: Correlated samples



김성훈 교수님 유튜브 (https://youtu.be/S1Y9eys2bdg)

Problems of the Naïve DGN 1: Correlated samples



Problems of the Naïve DGN 2: Non-stationary target

- Target 값도 계속 변화하여 Weight가 수렴하지 못함

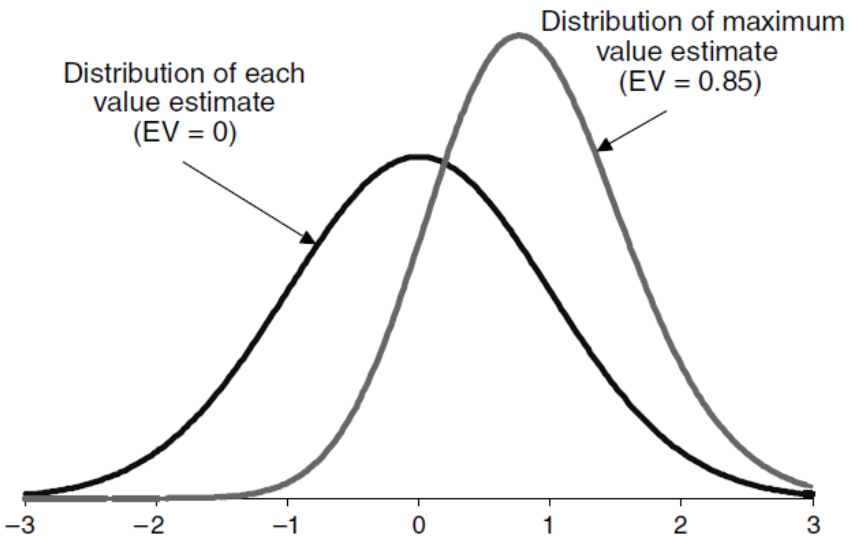
$$\text{minimize } L(\theta) = \|y - Q_\theta(s,a)\|_2^2$$

Target $y = r + \gamma \max_{a'} Q_\theta(s', a')$

Prediction $Q_\theta(s,a)$

Problems of the Naïve DGN 2: Non-stationary target

- Target 값도 계속 변화하여 Weight가 수렴하지 못함
- 추정된 Q값으로 새로운 Q값을 구하기때문에 Q-value가 과대평가

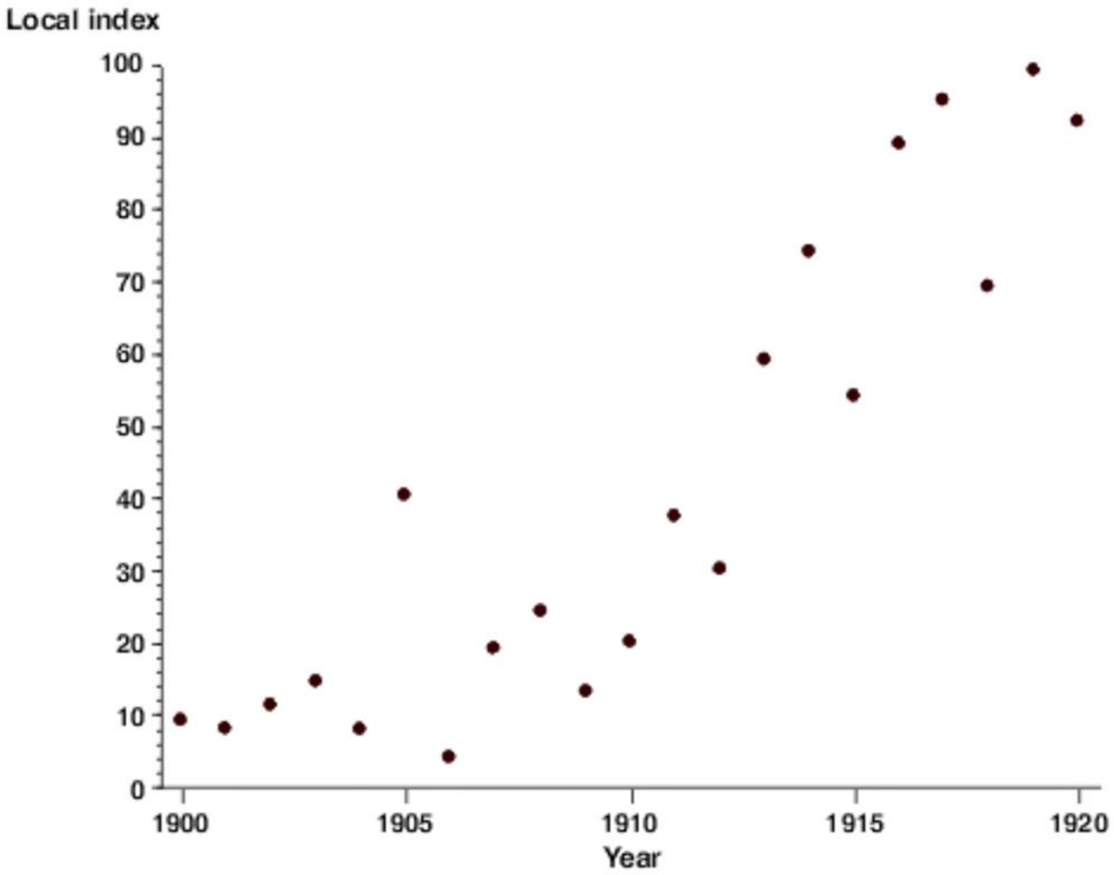


Solutions

Capture and replay

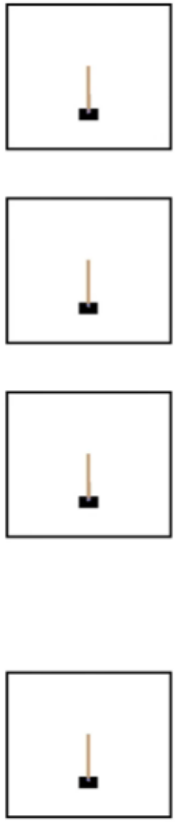
Separate target network

Problems of the Naïve DGN 1: Correlated samples



김성훈 교수님 유튜브 (https://youtu.be/S1Y9eys2bdg)

Solution 1: Capture and replay



Capture →

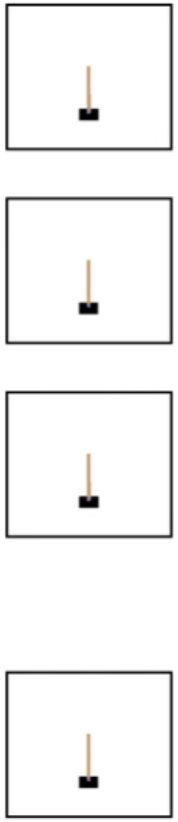
s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\dots$
$s_t, a_t, r_{t+1}, s_{t+1}$

random sample
& Replay →

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$

김성훈 교수님 유튜브 (https://youtu.be/S1Y9eys2bdg)

Solution 1: Capture and replay



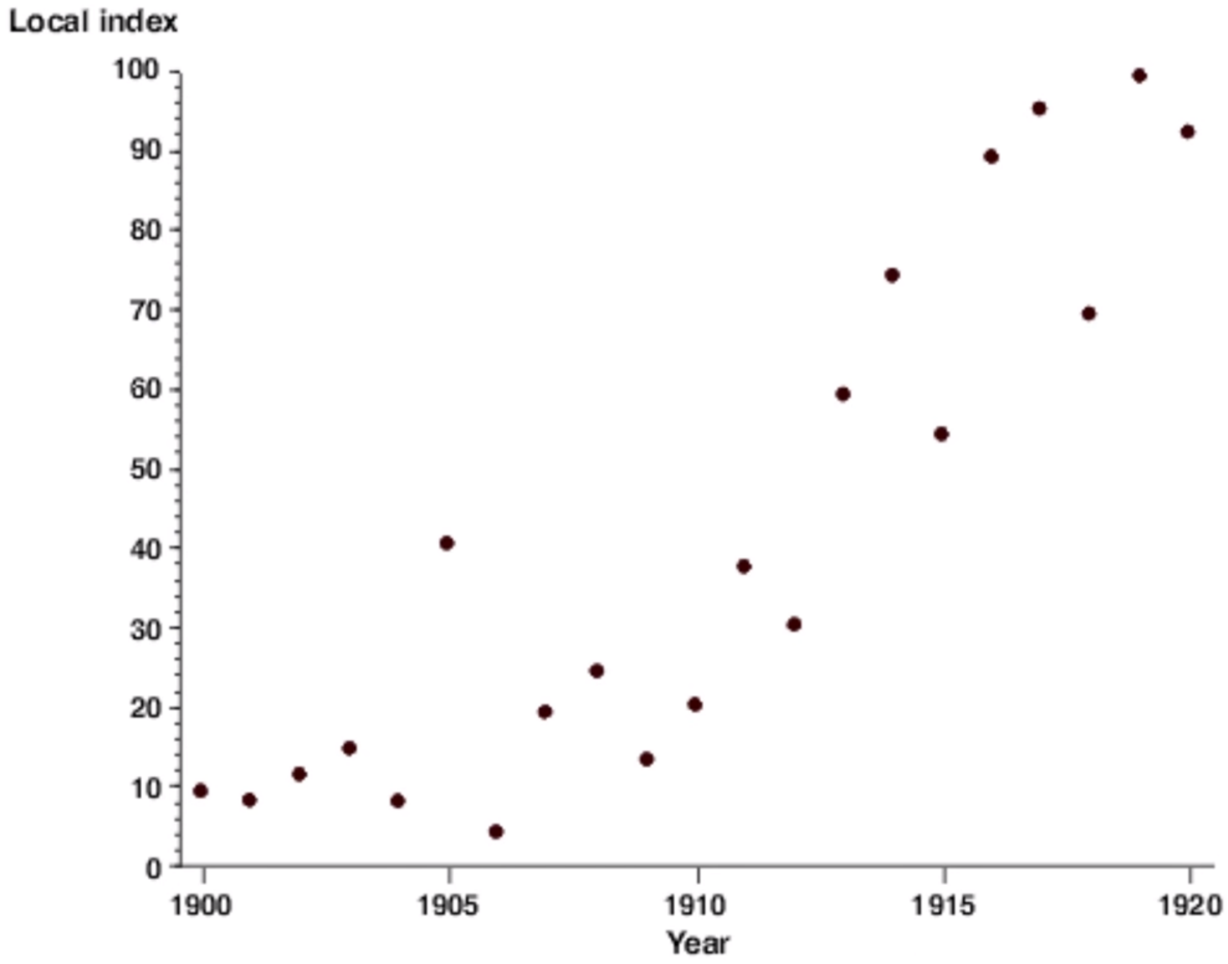
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Minih et al., Playing Atari with Deep Reinforcement Learning, Arxiv 2013
김성훈 교수님 유튜브 (https://youtu.be/S1Y9eys2bdg)

Solution 1: Capture and replay

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$



Problems of the Naïve DGN 2: Non-stationary target

- Target 값도 계속 변화하여 Weight가 수렴하지 못함

$$\text{minimize } L(\theta) = \left\| \left(r + \gamma \max_{a'} Q_\theta(s', a') \right) - Q_\theta(s, a) \right\|_2^2$$

Solution 2: separate target network

$$\text{minimize } L(\theta) = \left\| \left(r + \gamma \max_{a'} Q_\theta(s', a') \right) - Q_\theta(s, a) \right\|_2^2$$

$$\Downarrow$$

$$\text{minimize } L(\theta) = \left\| \left(r + \gamma \max_{a'} Q_{\theta'}(s', a') \right) - Q_\theta(s, a) \right\|_2^2$$

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

Initialize the replay memory and two identical Q approximators (DNN). $\hat{Q}$ is our target approximator.

For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do** ← Play m episodes (full games)
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    For $t = 1$,T **do**
        With probability ε select a random action a_t
        otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
        Every C steps reset $\hat{Q} = Q$
    End For
End For

Start episode from x_1 (pixels at the starting screen).
Preprocess the state (include 4 last frames, RGB to grayscale conversion, downsampling, cropping)

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do** ← For each time step during the episode
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    $\left.\begin{array}{l}\text{With probability } \varepsilon \text{ select a random action } a_t \\ \text{otherwise select } a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)\end{array}\right\}$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

With small probability select a random action (explore), otherwise select the, currently known, best action (exploit).

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Execute the chosen action and store the (processed) observed transition in the replay memory

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Experience replay:
Sample a random minibatch of transitions from replay memory and perform gradient decent step on Q (not on $\hat{Q}$)

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$ ← Once every several steps set the target function, $\hat{Q}$, to equal Q
  End For
End For

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Such delayed online learning helps in practice:
"This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $Q(s_t, a_t)$ often also increases $Q(s_{t+1}, a)$ for all a and hence also increases
the target y_j, possibly leading to oscillations or divergence of the policy

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

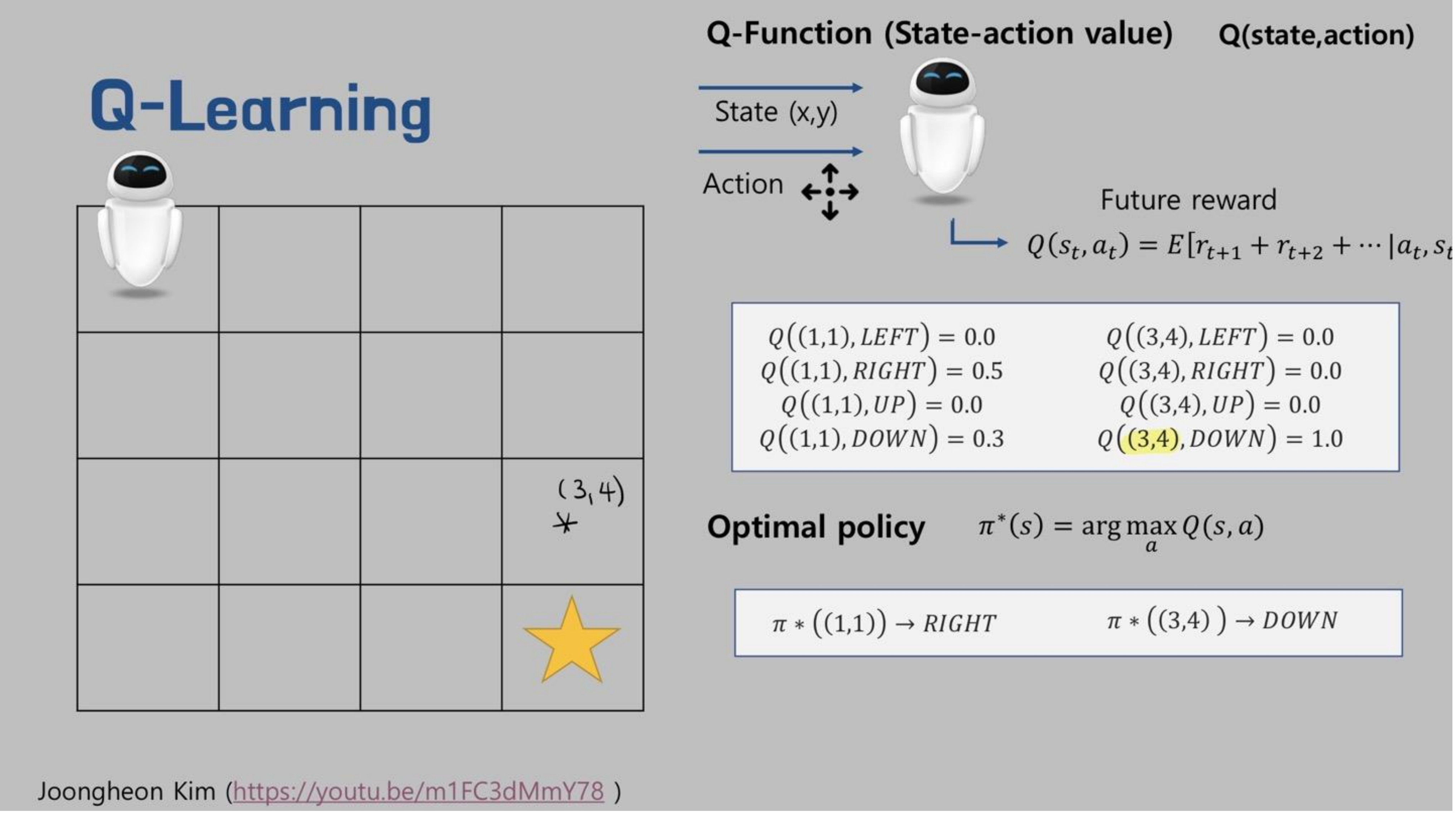
Conclusion

- Naïve Q-learning: Q-table를 Neural Network로 대체
  - Non-stationary target, Correlated samples문제가 있음
- Solutions
  - Capture and replay
  - Separate target network

References

- Deep Reinforcement Learning, UW CSE Deep Learning – Felix Leeb
- CSCE-689 Reinforcement Learning, Guni Sharon
- Lecture 7: DQN (Sung Kim) https://youtu.be/S1Y9eys2bdg
- Mnih et al., Human-level control through deep reinforcement learning, Nature 2015
- Mnih et al., Playing Atari with Deep Reinforcement Learning, Arxiv 2013

오타수정



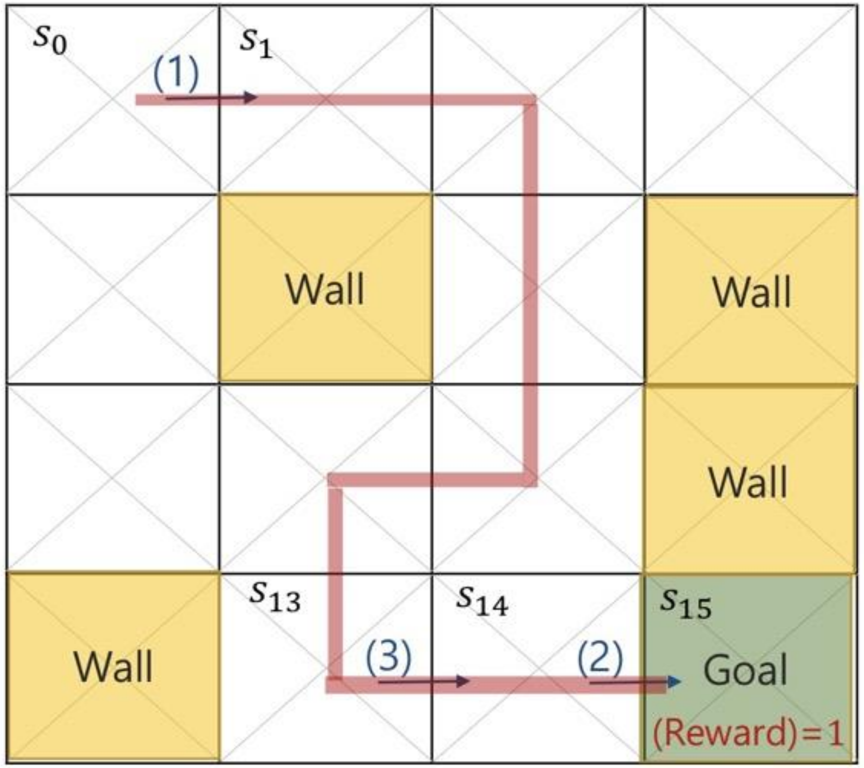
state(x,y)이면 Q((3,4),DOWN) =1.0 에서 state가 (3,4)가 아닌 (4,3) 아닌가요 ? 저 로봇이 (1,1)에 있다고하면 x축 방향으로 3칸, y축 방향으로 2칸 가서 (1+3, 1+2) 해서 (4,3) 같은데 왜 (3,4)인지 모르겠습니다.

밑에 슬라이드에서도 case 3에서 max[0,0,1,0] 이면 (U,D,L,R)니까 left로 가는것 아닌가요..?
Goal로 갈려면 s14에서 right로 가야하므로 max[0,0,0,1] 이 되어야 할 것같은데 왜 [0,0,1,0] 인지 모르겠습니다.

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

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)