Day 2. Deep Q-Network

SAMSUNG AI

Reinforcement Learning

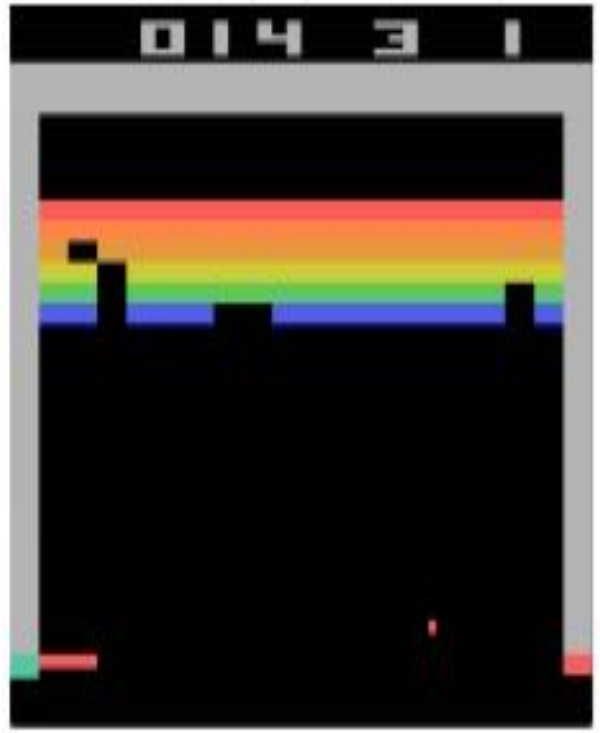
June 17, 2022

Jaeuk Shin, Mingyu Park

CORE
Control + Optimization Research Lab

Deep Q-Network : Why so Successful?

Is DQN merely a Q-learning algorithm with a neural network?



DQN agent playing Breakout

Two important techniques used in DQN:

1. use of **experience replay**
    - increase sample efficieny
2. use of **target Q-network**
    - mitigate training stability

$\Longrightarrow$ We will see how these features are implemented in practice!

Deep Q-Network : Implementation

Goal. Use a **neural network** to approximate a Q-function! (How?)

If there are m possible actions, there are two options:

1. convert a_j into **one-hot vector** $\mathbf{e}_j$, and feed $(s, \mathbf{e}_j)$ to a network which returns $Q(s, a_j; \theta)$

2. network returns m Q-values:

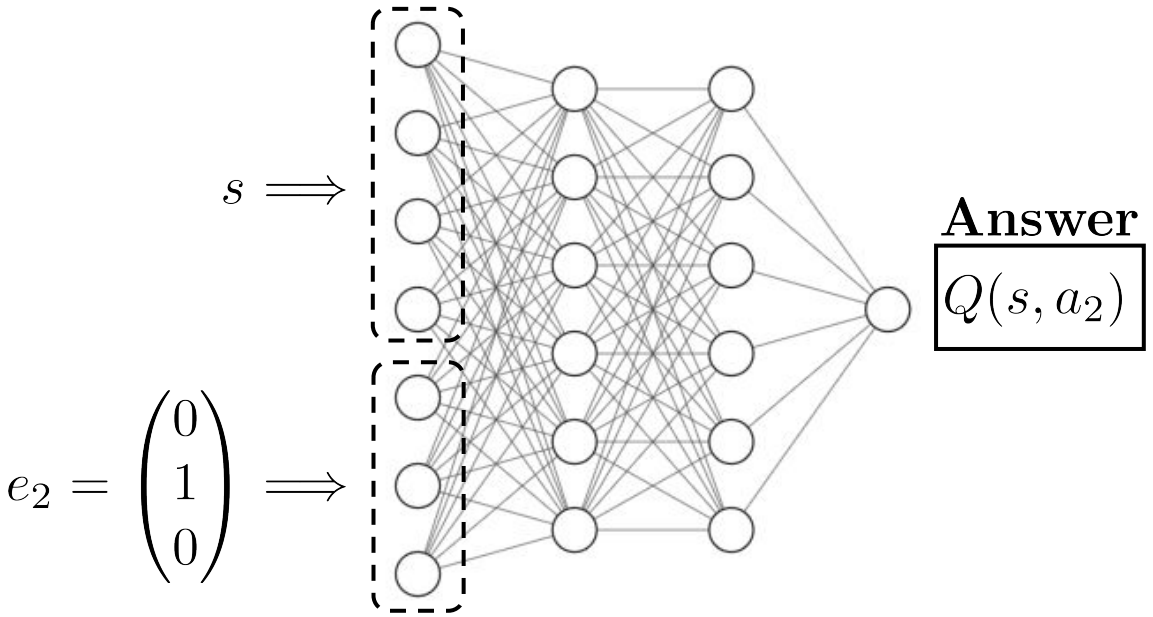
$$Q(s;\theta) = (Q(s, a_0; \theta), Q(s, a_1; \theta), \cdots Q(s, a_{m-1}; \theta))^\top.$$

Deep Q-Network : Implementation

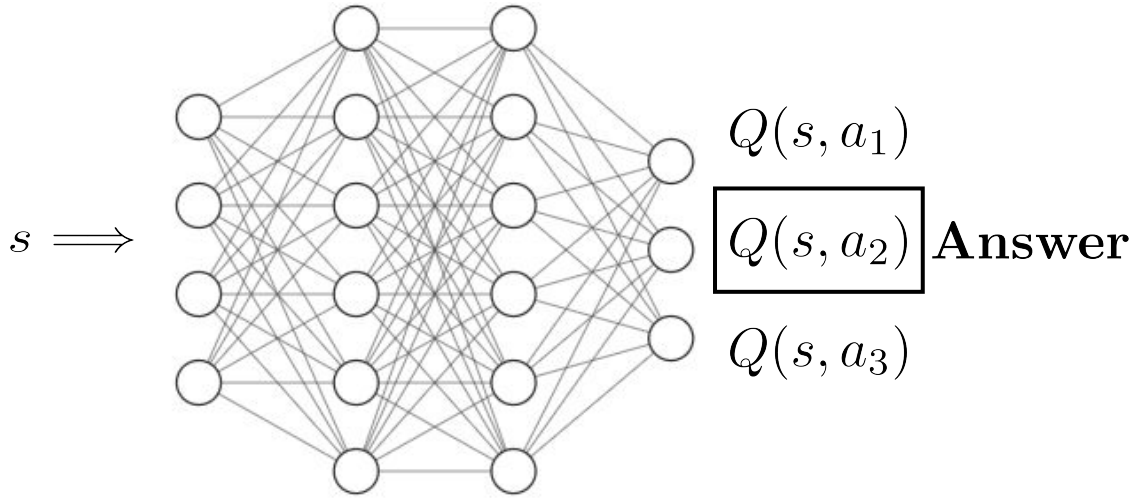
ex. 4 dim. state input & 3 actions

Query 1. Given a state s, What is $Q(s, a_2)$?

1. using one-hot vector encoding:



2. using multiple output architecture:

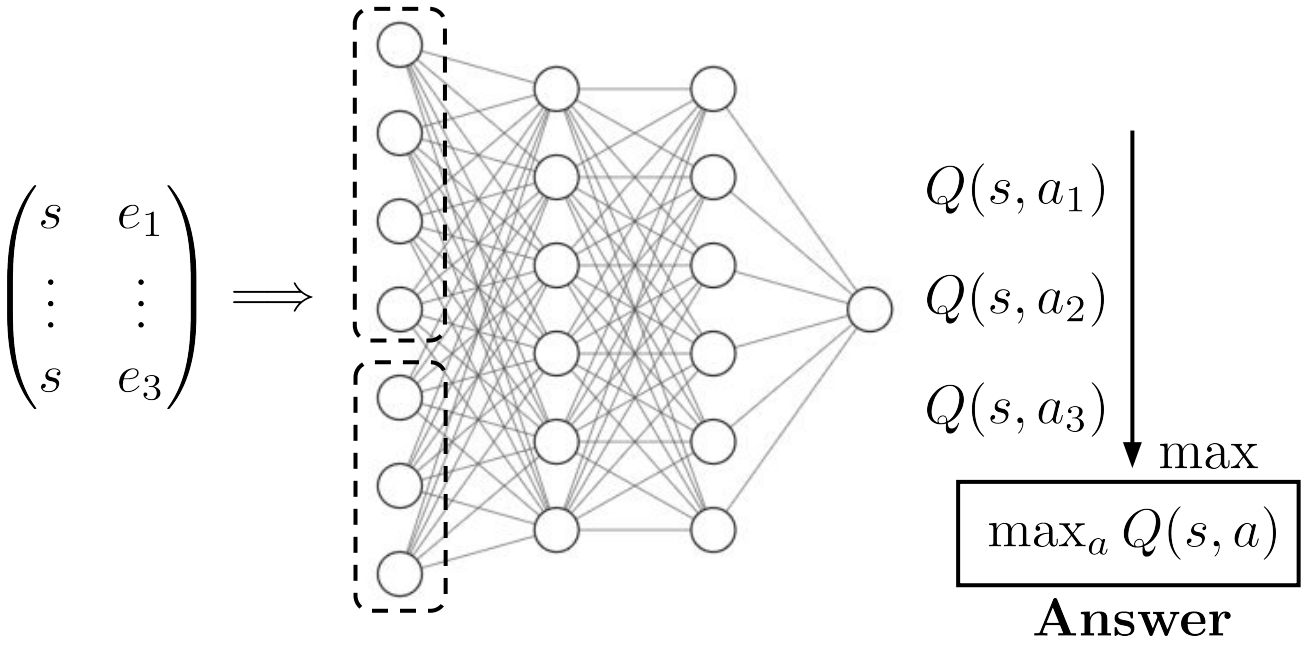


Deep Q-Network : Implementation

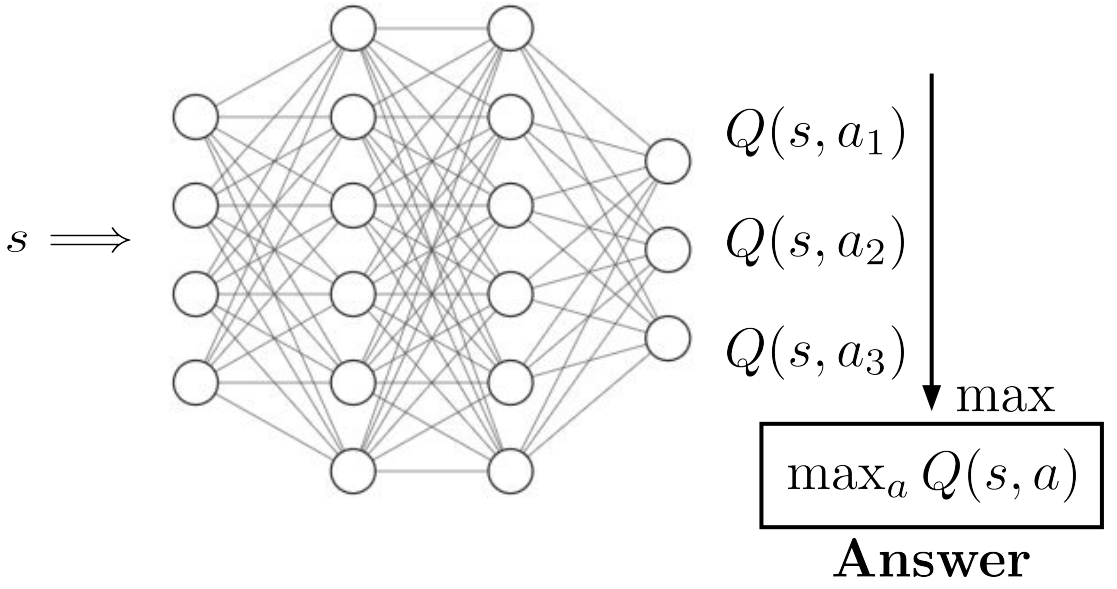
ex. 4 dim. state input & 3 actions

Query 2. Given a state s, What is $\max_a Q(s,a)$?

1. using one-hot vector encoding:



2. using multiple output architecture:



CORE
Control + Optimization Research Lab

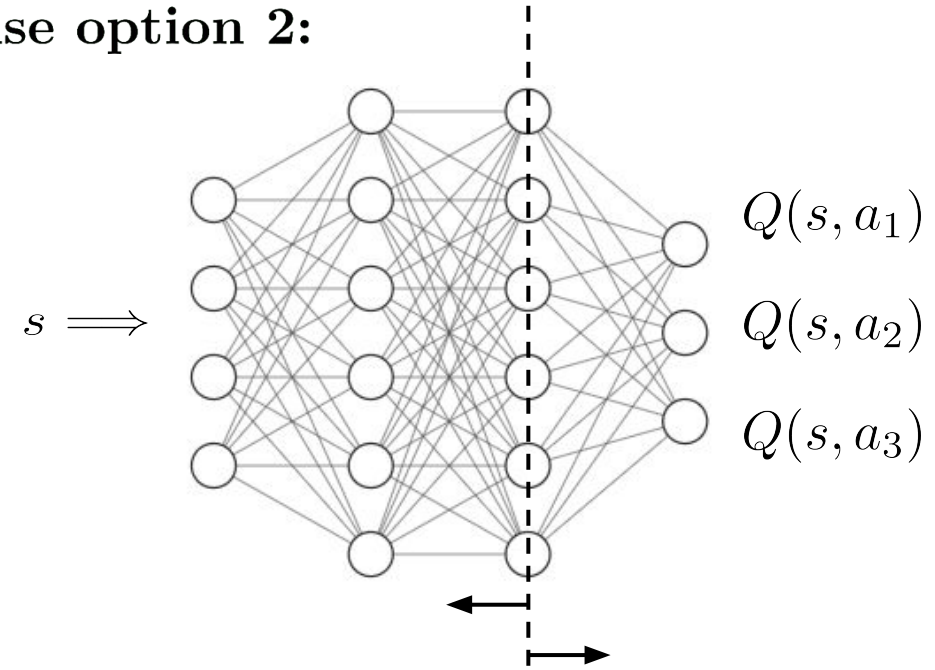
Deep Q-Network : Implementation

ex. 4 dim. state input & 3 actions

Query 3. Given a batch $s_1, \cdots, s_{256}$, What is $\max_a Q(s_1, a), \cdots, \max_a Q(s_{256}, a)$?

$\Longrightarrow$ Definitely, option 2 is better in this case.

In practice, we use option 2:



What if action is continuous?

Deep Q-Network : Implementation

2-layer neural network in PyTorch

```
1   class Critic(nn.Module):
2       def __init__(self, state_dim, num_action, hidden_size1, hidden_size2):
3           super(Critic, self).__init__()
4           self.fc1 = nn.Linear(state_dim, hidden_size1)
5           self.fc2 = nn.Linear(hidden_size1, hidden_size2)
6           self.fc3 = nn.Linear(hidden_size2, num_action)
7
8       def forward(self, state):
9           x = F.relu(self.fc1(state))
10          x = F.relu(self.fc2(x))
11          q = self.fc3(x)
12          return q
```

neural network parameters

Given s, compute a vector of Q-values.

other non-linearities : tanh, etc.

Deep Q-Network : Implementation

```python
class DQNAgent:
    def __init__(self, obs_dim, num_act, hidden1, hidden2):
        self.obs_dim = obs_dim
        self.num_act = num_act
        self.critic = Critic(obs_dim, num_act, hidden1, hidden2).to(device)

    def act(self, state, epsilon=0.0):
        if np.random.rand() < epsilon:
            return np.random.randint(self.num_act)
        else:
            self.critic.eval()
            s = torch.Tensor(state).view(1, self.obs_dim).to(device)
            q = self.critic(s)
            return np.argmax(q.cpu().detach().numpy())
```

ε-greedy method

Deep Q-Network : Implementation

```
1   def update(agent, replay_buf, gamma, critic_optim, target_critic, tau, batch_size):
2       agent.critic.train()
3       batch = replay_buf.sample_batch(batch_size)
4
5       with torch.no_grad():
6           observations = torch.Tensor(batch['state']).to(device)
7           actions = torch.tensor(batch['action'], dtype=torch.long).to(device)
8           rewards = torch.Tensor(batch['reward']).to(device)
9           next_observations = torch.Tensor(batch['next_state']).to(device)
10          terminals = torch.Tensor(batch['done']).to(device)
11          mask = 1.0 - terminals
12
13          next_q = torch.unsqueeze(target_critic(next_observations).max(1)[0], 1)
14          target = rewards + gamma * mask * next_q
15
16      out = agent.critic(observations).gather(1, actions)
17
18      loss_ftn = MSELoss()
19      loss = loss_ftn(out, target)
20      critic_optim.zero_grad()
21      loss.backward()
22      critic_optim.step()
23
24      for p, targ_p in zip(agent.critic.parameters(), target_critic.parameters()):
25          targ_p.data.copy_((1. - tau) * targ_p + tau * p)
26      return
```

unroll batch

training target construction

$$y_j = r_j + \gamma \max_{a'} Q\left(s'_j, a; \theta^-\right)$$

target network

mean squared loss:

$$\frac{1}{N}\sum_{j=1}^{N}(Q(s_j, a_j; \theta) - y_j)^2$$

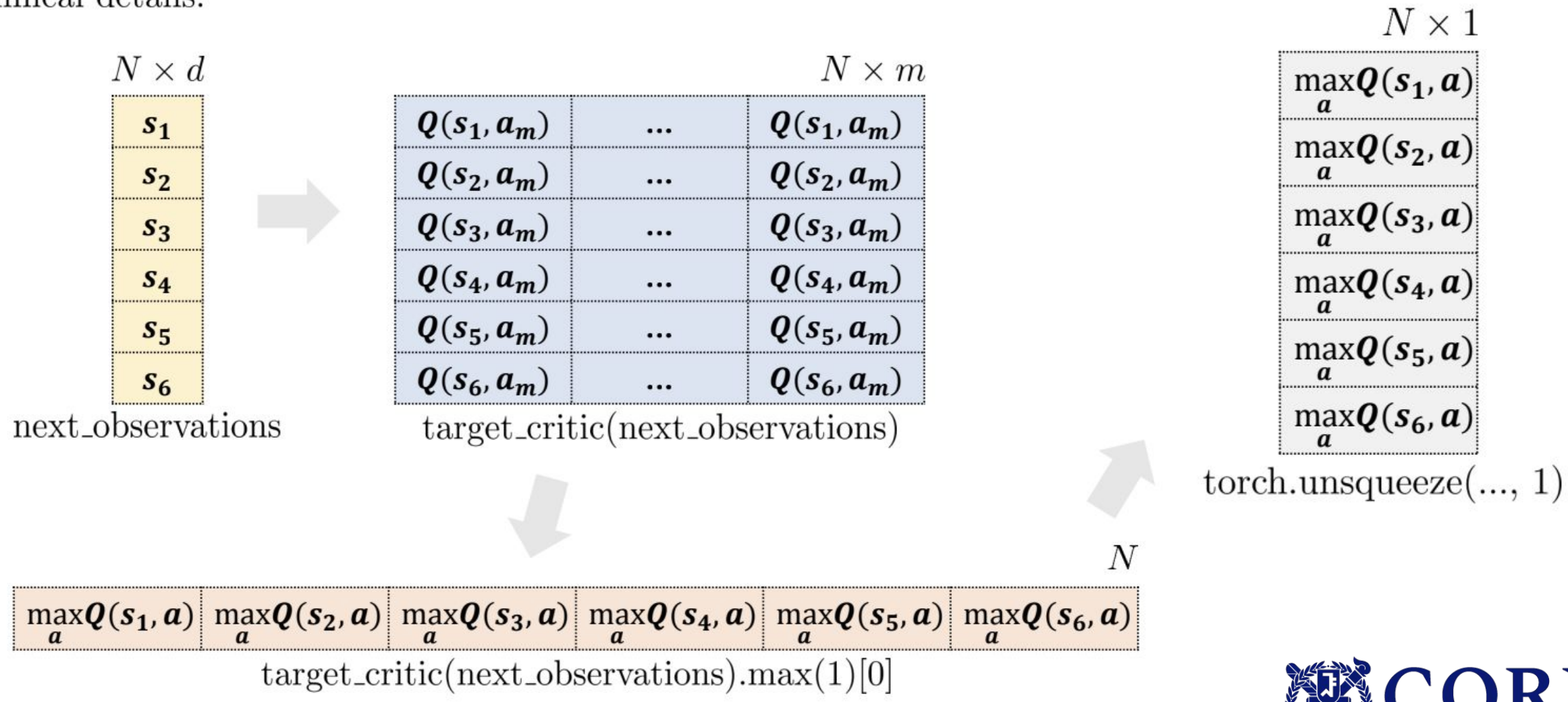
gradient descent

update target network

$$\theta^- \leftarrow (1-\tau)\theta^- + \tau\theta$$

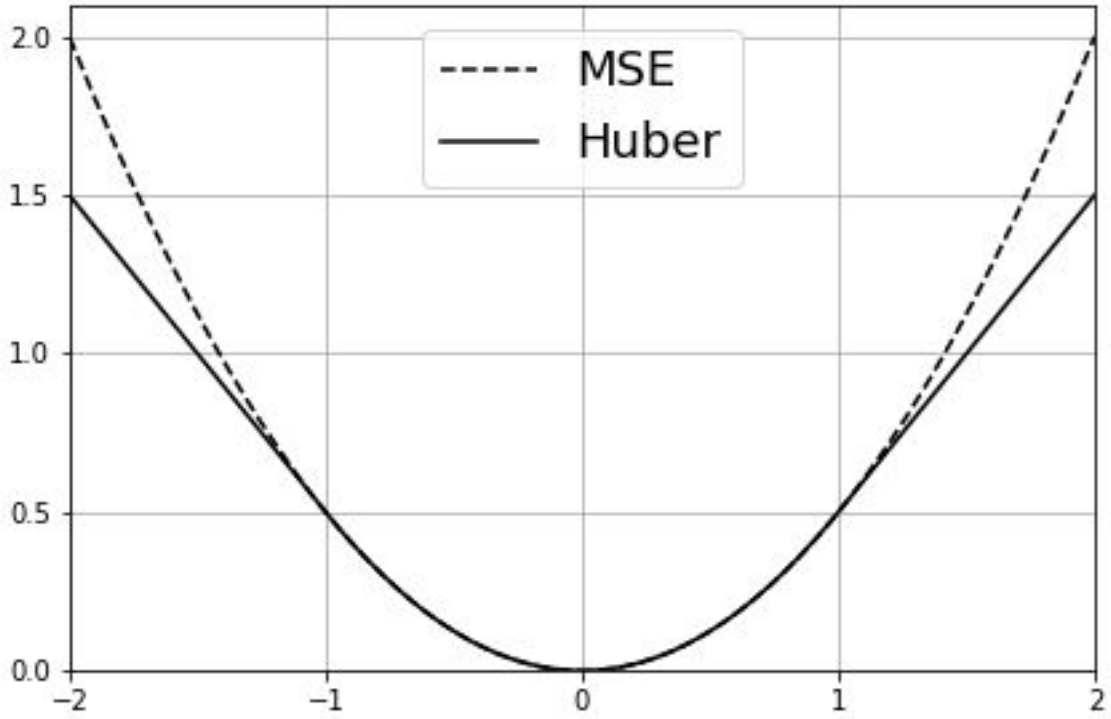
Deep Q-Network : Implementation

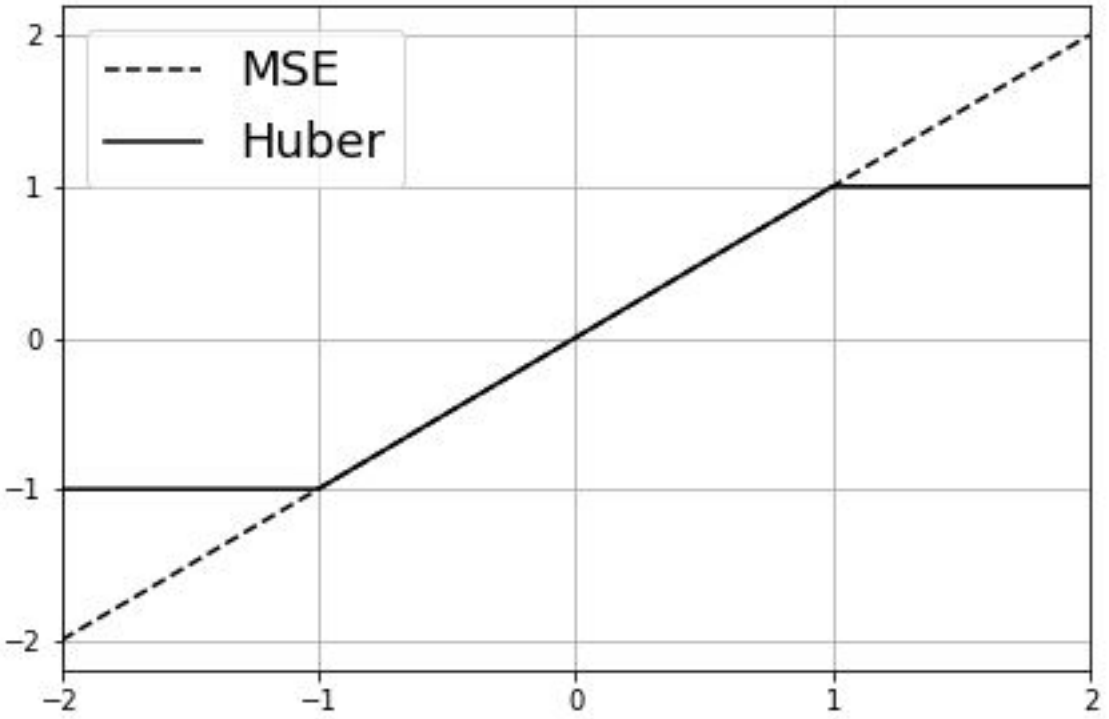
technical details:



Deep Q-Network : Implementation

Alternative choice of loss function? $\Rightarrow$ **Huber loss**





take gradient

Deep Q-Network : Example

OpenAI Gym CartPole-v1

$$s := (x, \dot{x}, \theta, \dot{\theta}) \quad \begin{cases} x & : \text{x coordinate of the cart} \\ \dot{x} & : \text{x speed of the cart} \\ \theta & : \text{Angle of the pole} \\ \dot{\theta} & : \text{Angular velocity of the pole} \end{cases}$$

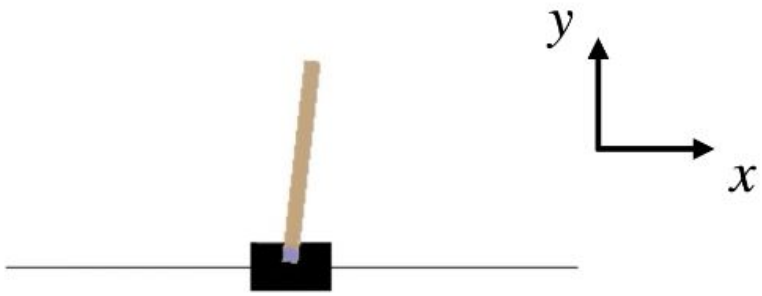


Figure 3: Illustration of the cartpole problem. $\theta = 0$ is set to the $+y$ direction.

Let's try it!

Deep Q-Network : Further Improvement?

Limitation?

- stability of training
- sample efficiency
- overestimation bias
- exploration scheme

Key question) How to deal with continuous action space?

How to overcome these issues?

1. use variants of DQN, e.g., double DQN (Hasselt, '15), prioritized experience replay (Schaul, '15)
2. combine it with policy gradient method $\Longrightarrow$ **actor-critic algorithms**