

Double DQN (DDQN)

이동민

삼성전자 서울대 공동연구소

Jul 16, 2019

Outline

- Environment : CartPole
- Points to note when implementing RL algorithms
- Double DQN (DDQN)
- DQN vs. DDQN - Learning curve

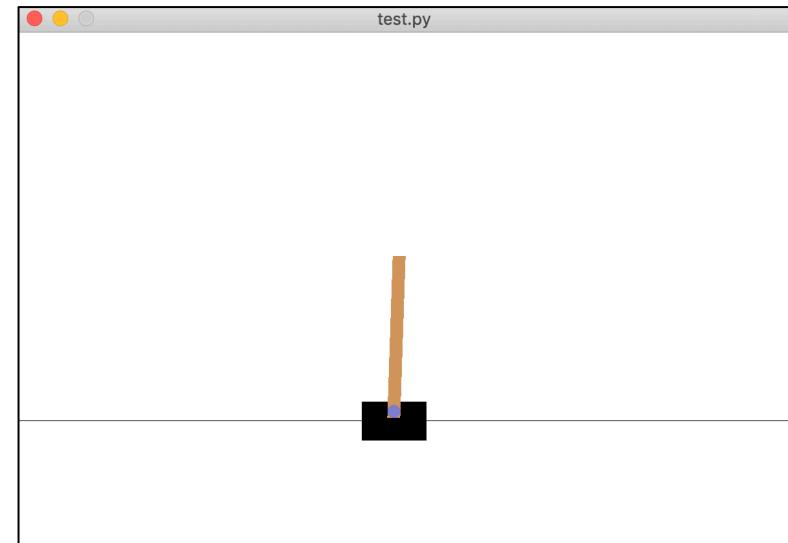
CartPole

- Env name : CartPole-v1
- States : Continuous observation spaces

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	$\sim -41.8^\circ$	$\sim 41.8^\circ$
3	Pole Velocity At Tip	-Inf	Inf

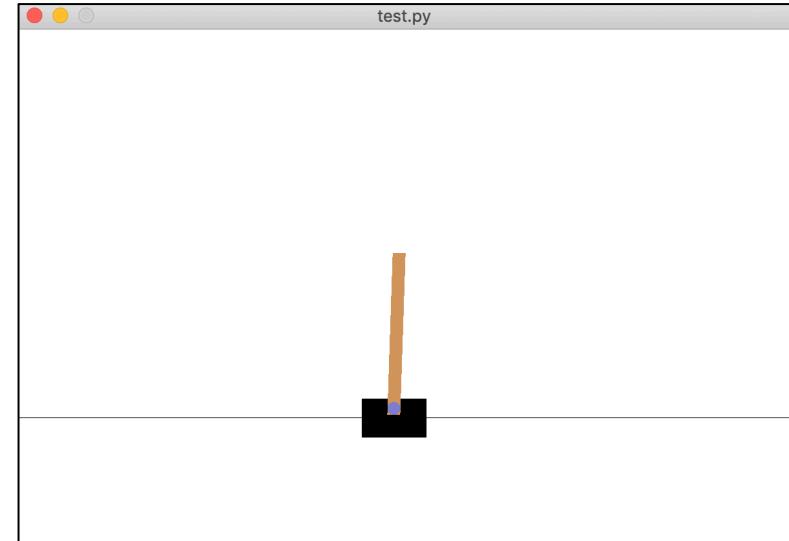
- Actions : Discrete action spaces

Num	Action
0	Push cart to the left
1	Push cart to the right



CartPole

- Reward
 - Reward is 1 for every step taken, including the termination step
- Episode Termination
 - Pole Angle is more than $\pm 12^\circ$
 - Cart Position is more than ± 2.4
 - Episode length is greater than 500



CartPole

- Test code

```
import gym

env = gym.make('CartPole-v1')

for episode in range(10000): ← Episode
    done = False
    state = env.reset()

    while not done: ← Step
        env.render()

        action = env.action_space.sample()
        next_state, reward, done, _ = env.step(action)

        print('state: {} | action: {} | next_state: {} | reward: {} | done: {}'.format(
            state, action, next_state, reward, done))

        state = next_state

    if done:
        break
```

state: [0.02882383 0.53942364 -0.06136833 -0.95704334] | action: 1 |
next_state: [0.0396123 0.73531478 -0.0805092 -1.26835788] | reward:
1.0 | done: False
state: [0.0396123 0.73531478 -0.0805092 -1.26835788] | action: 1 |
next_state: [0.0543186 0.93136736 -0.10587636 -1.58512834] | reward:
1.0 | done: False
state: [0.0543186 0.93136736 -0.10587636 -1.58512834] | action: 0 |
next_state: [0.07294594 0.73765175 -0.13757892 -1.32725157] | reward:
1.0 | done: False

Points to note when implementing RL algorithms

- 우선적으로 **환경**이 잘 동작하는지 random agent를 통해 확인해야 한다.
 - State, action, reward, next state, done 출력해보기
 - 각 action이 어떤 action인지, 어떤 값인지를 확인하기
- Tensor의 **shape**을 항상 주의 깊게 봐야한다. (**제일 중요!**)
- Tensor의 **자료형**이 현재 일반 list인지, numpy의 array인지, torch의 tensor인지를 잘 확인해야한다.

Double DQN (DDQN)

- DQN (Final version) + DDQN

Algorithm 1: deep 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

$$L^{DoubleQ} = (r + \gamma \max_{a'} Q_{\theta^-}(s', \text{argmax}_{a'} Q_{\theta}(s', a')) - Q_{\theta}(s, a))^2$$

$$L^Q = (r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_{\theta}(s, a))^2$$

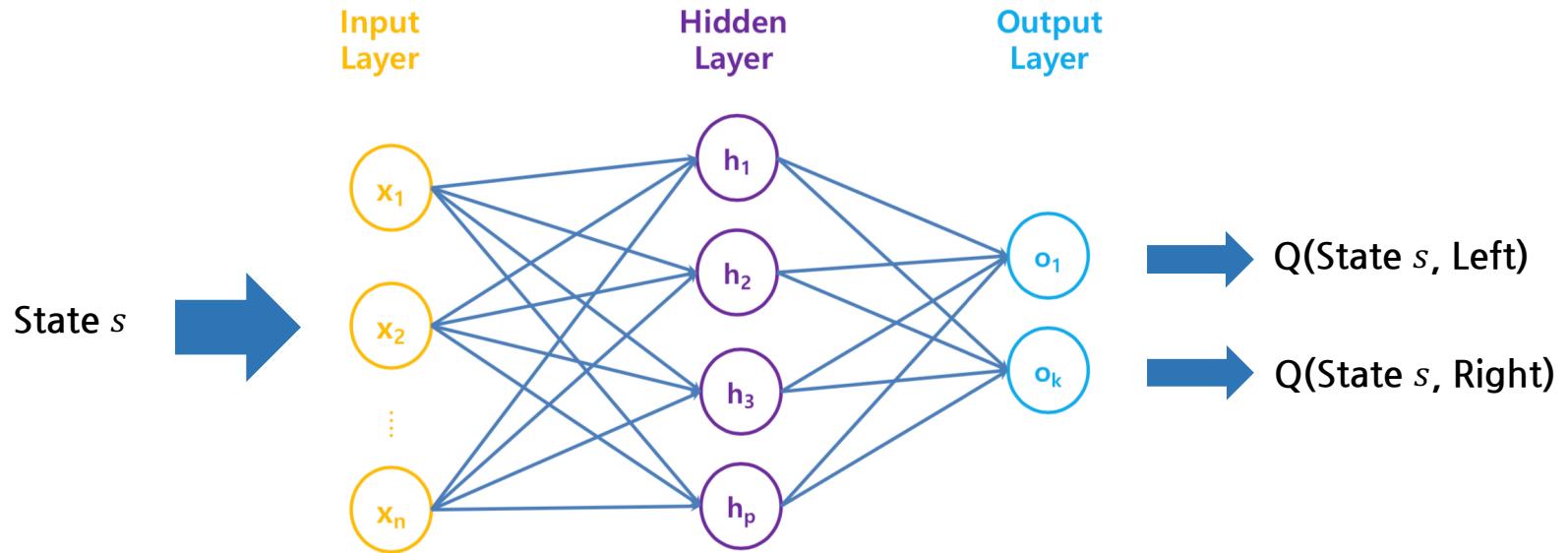
source : <https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>

Double DQN (DDQN)

- Learning process
 1. 상태에 따른 행동 선택
 2. 환경에서 선택한 행동으로 한 time step을 진행한 후, 다음 상태와 보상을 받음
 3. Sample (s, a, r, s') 을 replay buffer에 저장
 4. Replay buffer에서 랜덤으로 sample을 추출
 5. 추출한 sample로 학습
 6. 일정한 step마다 target model 업데이트

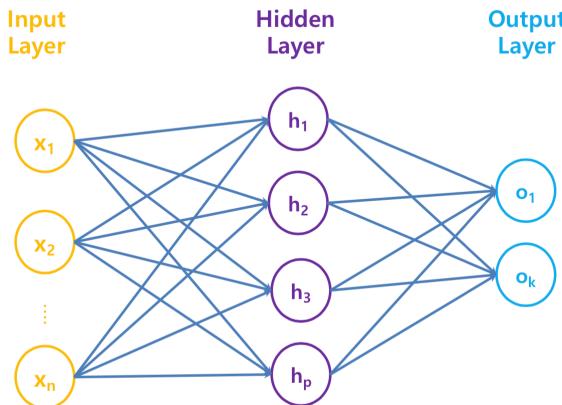
Double DQN (DDQN)

- Q-Network (Function approximator)



Double DQN (DDQN)

- Q-Network (Function approximator)



```
1 import torch
2 import torch.nn as nn
3
4 class QNet(nn.Module):
5     def __init__(self, state_size, action_size, args):
6         super(QNet, self).__init__()
7         self.fc1 = nn.Linear(state_size, args.hidden_size)
8         self.fc2 = nn.Linear(args.hidden_size, action_size)
9
10    def forward(self, x):
11        x = torch.tanh(self.fc1(x))
12        q_values = self.fc2(x)
13        return q_values
```

Learning process

1. 상태에 따른 행동 선택

```
110     q_values = q_net(torch.Tensor(state))  
111     action = get_action(q_values, action_size, args.epsilon)
```

- ϵ -greedy policy (Exploitation vs. Exploration)

$$\pi(s) = \begin{cases} \underset{a}{\operatorname{argmax}} Q(s, a), & 1 - \epsilon \\ a \text{ random action}, & \epsilon \end{cases}$$

```
64     def get_action(q_values, action_size, epsilon):  
65         if np.random.rand() <= epsilon:  
66             return random.randrange(action_size)  
67         else:  
68             _, action = torch.max(q_values, 1)  
69             return action.numpy()[0]
```

- ϵ decay = 0.00005 (Initial value ϵ = 1, Initial exploration = 1000)

```
124     if steps > args.initial_exploration:  
125         args.epsilon -= args.epsilon_decay  
126         args.epsilon = max(args.epsilon, 0.1)
```

Learning process

2. 환경에서 선택한 행동으로 한 time step을 진행한 후, 다음 상태와 보상을 받음

```
113 |     next_state, reward, done, _ = env.step(action)
```

3. Sample (s, a, r, s') 을 replay buffer에 저장

```
93 |     replay_buffer = deque(maxlen=10000)  
  
117 |     mask = 0 if done else 1  
118 |  
119 |     replay_buffer.append((state, action, reward, next_state, mask))
```

4. Replay buffer에서 랜덤으로 sample을 추출 (Batch size : 32)

```
128 |     mini_batch = random.sample(replay_buffer, args.batch_size)
```

Learning process

5. 추출한 sample로 학습

- MSE Loss

$$L^{DoubleQ} = \underbrace{(r + \gamma Q_{\theta^-}(s', \operatorname{argmax}_{a'} Q_{\theta}(s', a')) - Q_{\theta}(s, a))^2}_{\text{Target}} \quad \underbrace{\phantom{(r + \gamma Q_{\theta^-}(s', \operatorname{argmax}_{a'} Q_{\theta}(s', a')) - Q_{\theta}(s, a))}^2}_{\text{Prediction}}$$

```
45     criterion = torch.nn.MSELoss()
46
47     # get Q-value
48     q_values = q_net(torch.Tensor(states))
49     q_value = q_values.gather(1, actions.unsqueeze(1)).view(-1)
```

- Gather - gather(dim, index)

```
tensor = torch.Tensor([[1,3], [3,2], [5,4], [1,4], [4,2]])
index = torch.LongTensor([[0], [1], [0], [0], [1]])

tensor = tensor.gather(1, index)
print(tensor)
...
tensor([[1.],
       [2.],
       [5.],
       [1.],
       [2.]])
...
```

Learning process

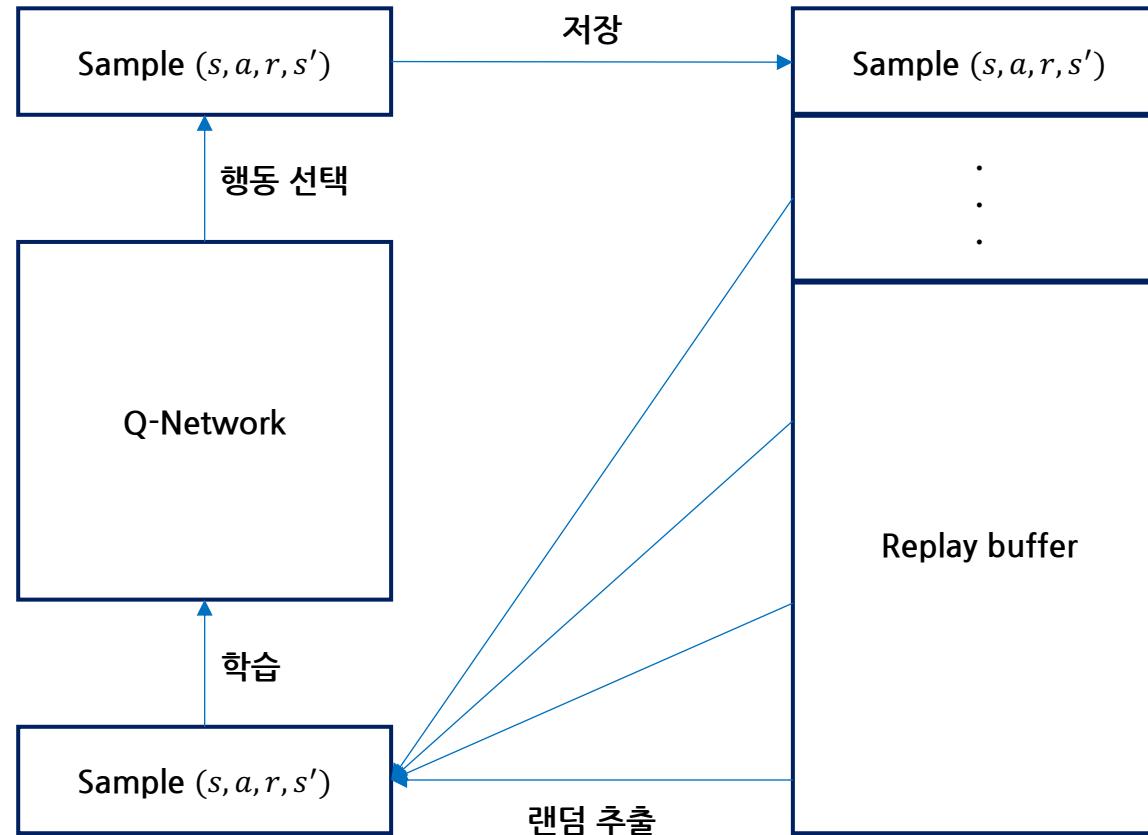
5. 추출한 sample로 학습

- MSE Loss

$$L^{DoubleQ} = \frac{(r + \gamma Q_{\theta^-}(s', \underset{a'}{\operatorname{argmax}} Q_{\theta}(s', a')) - Q_{\theta}(s, a))^2}{\text{Target} \quad \text{Prediction}}$$

```
45 criterion = torch.nn.MSELoss()
46
47 # get Q-value
48 q_values = q_net(torch.Tensor(states))
49 q_value = q_values.gather(1, actions.unsqueeze(1)).view(-1)
50
51 # get target
52 next_q_values = q_net(torch.Tensor(next_states))
53 next_q_value_index = next_q_values.max(1)[1]
54
55 target_next_q_values = target_q_net(torch.Tensor(next_states))
56 target_next_q_value = target_next_q_values.gather(1, next_q_value_index.unsqueeze(1)).view(-1)
57 target = rewards + masks * args.gamma * target_next_q_value
58
59 loss = criterion(q_value, target.detach())
60 optimizer.zero_grad()
61 loss.backward()
62 optimizer.step()
```

Learning process



Learning process

6. 일정한 step마다 target model 업데이트

- Initialize target model

```
89 |     update_target_model(q_net, target_q_net)
```

```
71  def update_target_model(net, target_q_net):  
72      target_q_net.load_state_dict(net.state_dict())
```

- Target model 업데이트 (Update target : 100 step)

```
133 |             if steps % args.update_target:  
134 |                 update_target_model(q_net, target_q_net)
```

Hyperparameter

```
14 parser = argparse.ArgumentParser()
15 parser.add_argument('--env_name', type=str, default="CartPole-v1")
16 parser.add_argument('--load_model', type=str, default=None)
17 parser.add_argument('--save_path', default='./save_model/', help='')
18 parser.add_argument('--render', action="store_true", default=False)
19 parser.add_argument('--gamma', type=float, default=0.99)
20 parser.add_argument('--hidden_size', type=int, default=64)
21 parser.add_argument('--batch_size', type=int, default=32)
22 parser.add_argument('--initial_exploration', type=int, default=1000)
23 parser.add_argument('--epsilon', type=float, default=1.0)
24 parser.add_argument('--epsilon_decay', type=float, default=0.00005)
25 parser.add_argument('--update_target', type=int, default=100)
26 parser.add_argument('--max_iter_num', type=int, default=1000)
27 parser.add_argument('--log_interval', type=int, default=10)
28 parser.add_argument('--goal_score', type=int, default=400)
29 parser.add_argument('--logdir', type=str, default='./logs',
30 |   |   |   |   |   help='tensorboardx logs directory')
31 args = parser.parse_args()
```

Main loop

- Initialization

- Seed - random number 고정
- Q-Network
- Target Q-Network
- Optimizer
- Target model
- TensorboardX
- Replay buffer

```
75     def main():
76         env = gym.make(args.env_name)
77         env.seed(500)
78         torch.manual_seed(500)
79
80         state_size = env.observation_space.shape[0]
81         action_size = env.action_space.n
82         print('state size:', state_size)
83         print('action size:', action_size)
84
85         q_net = QNet(state_size, action_size, args)
86         target_q_net = QNet(state_size, action_size, args)
87         optimizer = optim.Adam(q_net.parameters(), lr=0.001)
88
89         update_target_model(q_net, target_q_net)
90
91         writer = SummaryWriter(args.logdir)
92
93         replay_buffer = deque(maxlen=10000)
94         running_score = 0
95         steps = 0
```

Main loop

- Episode 진행
 - Reshape state vector (4) → (1,4)
 - 상태에 따른 행동 선택
 - 다음 상태와 보상을 받음
 - Reshape next state vector
 - Reward, mask 설정
 - Replay buffer에 저장

```
97     for episode in range(args.max_iter_num):  
98         done = False  
99         score = 0  
100  
101         state = env.reset()  
102         state = np.reshape(state, [1, state_size])  
103  
104         while not done:  
105             if args.render:  
106                 env.render()  
107  
108             steps += 1  
109  
110             q_values = q_net(torch.Tensor(state))  
111             action = get_action(q_values, action_size, args.epsilon)  
112  
113             next_state, reward, done, _ = env.step(action)  
114  
115             next_state = np.reshape(next_state, [1, state_size])  
116             reward = reward if not done or score == 499 else -1  
117             mask = 0 if done else 1  
118  
119             replay_buffer.append((state, action, reward, next_state, mask))  
120  
121             state = next_state  
122             score += reward
```

Main loop

- Episode 진행
 - Step 수가 1000보다 작으면 계속 random action을 선택
 - Step 수가 1000보다 크면
 - ϵ decay (Initial value $\epsilon = 1$)
 - Replay buffer에서 랜덤으로 32개의 sample을 추출 → Mini batch
 - Train model
 - 100 step마다 target model 업데이트
 - Running score 설정

```
124     if steps > args.initial_exploration:  
125         args.epsilon -= args.epsilon_decay  
126         args.epsilon = max(args.epsilon, 0.1)  
127  
128         mini_batch = random.sample(replay_buffer, args.batch_size)  
129  
130         q_net.train(), target_q_net.train()  
131         train_model(q_net, target_q_net, optimizer, mini_batch)  
132  
133         if steps % args.update_target:  
134             update_target_model(q_net, target_q_net)  
135  
136         score = score if score == 500.0 else score + 1  
137         running_score = 0.99 * running_score + 0.01 * score
```

Main loop

- Print & Visualize log
- Running score > 400
 - Save model
 - 학습 종료

0 episode	running_score: 0.31	epsilon: 1.00
10 episode	running_score: 2.37	epsilon: 1.00
20 episode	running_score: 4.44	epsilon: 1.00
30 episode	running_score: 6.04	epsilon: 1.00
40 episode	running_score: 7.83	epsilon: 1.00
50 episode	running_score: 9.19	epsilon: 0.99
60 episode	running_score: 10.81	epsilon: 0.98
70 episode	running_score: 11.89	epsilon: 0.96
80 episode	running_score: 12.68	epsilon: 0.95
90 episode	running_score: 13.73	epsilon: 0.94
100 episode	running_score: 14.36	epsilon: 0.93

```
139         if episode % args.log_interval == 0:
140             print('{0} episode | running_score: {1:.2f} | epsilon: {2:.2f}'.format(
141                 episode, running_score, args.epsilon))
142             writer.add_scalar('log/score', float(score), episode)
143
144         if running_score > args.goal_score:
145             if not os.path.isdir(args.save_path):
146                 os.makedirs(args.save_path)
147
148             ckpt_path = args.save_path + 'model.pth'
149             torch.save(q_net.state_dict(), ckpt_path)
150             print('Running score exceeds 400. So end')
151             break
```

Train model

- Mini batch → Numpy array
 - mini_batch - (32, 5)

```
33     def train_model(q_net, target_q_net, optimizer, mini_batch):  
34         mini_batch = np.array(mini_batch)
```

```
[array([[ 0.06638455,  0.44188166, -0.05941764, -0.71998114]]) 0 1.0  
 array([[ 0.07522218,  0.24762993, -0.07381726, -0.44657625]]) 1]
```

```
[array([[ 0.03859636,  0.54112043, -0.05626789, -0.85790688]]) 0 1.0  
 array([[ 0.04941877,  0.34680833, -0.07342603, -0.58343404]]) 1]
```

```
[array([[ 0.11499082,  0.93411447, -0.1403067 , -1.50586197]]) 1 1.0  
 array([[ 0.13367311,  1.13063172, -0.17042394, -1.83885608]]) 1]
```

```
[array([[ 0.03836089, -0.0380637 , -0.03338654, -0.04749324]]) 0 1.0  
 array([[ 0.03759962, -0.2326914 , -0.03433641,  0.2344718 ]]) 1]
```

```
[array([[-0.11449923,  0.0169064 ,  0.15822003,  0.29739237]]) 1 1.0  
 array([[-0.1141611 ,  0.20946075,  0.16416787,  0.05849141]]) 1]
```

Train model

- Mini batch → Numpy array
 - mini_batch - (32, 5)
- Mini batch에 있는 32개의 sample들을 각각 나눔
 - states - (32, 4)
 - next_states - (32, 4)
- List → Torch tensor
 - actions - (32)
 - rewards - (32)
 - masks - (32)

```
33     def train_model(q_net, target_q_net, optimizer, mini_batch):  
34         mini_batch = np.array(mini_batch)  
35         states = np.vstack(mini_batch[:, 0])  
36         actions = list(mini_batch[:, 1])  
37         rewards = list(mini_batch[:, 2])  
38         next_states = np.vstack(mini_batch[:, 3])  
39         masks = list(mini_batch[:, 4])  
40  
41         actions = torch.LongTensor(actions)  
42         rewards = torch.Tensor(rewards)  
43         masks = torch.Tensor(masks)
```

Train model

- MSE Loss

$$L^{DoubleQ} = \frac{(r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a))^2}{\text{Target} \quad \quad \quad \text{Prediction}}$$

- cf) $L^Q = (r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a))^2$

```
45     criterion = torch.nn.MSELoss()
46
47     # get Q-value
48     q_values = q_net(torch.Tensor(states))
49     q_value = q_values.gather(1, actions.unsqueeze(1)).view(-1)
50
51     # get target
52     next_q_values = q_net(torch.Tensor(next_states))
53     next_q_value_index = next_q_values.max(1)[1]
54
55     target_next_q_values = target_q_net(torch.Tensor(next_states))
56     target_next_q_value = target_next_q_values.gather(1, next_q_value_index.unsqueeze(1)).view(-1)
57     target = rewards + masks * args.gamma * target_next_q_value
58
59     loss = criterion(q_value, target.detach())
60     optimizer.zero_grad()
61     loss.backward()
62     optimizer.step()
```

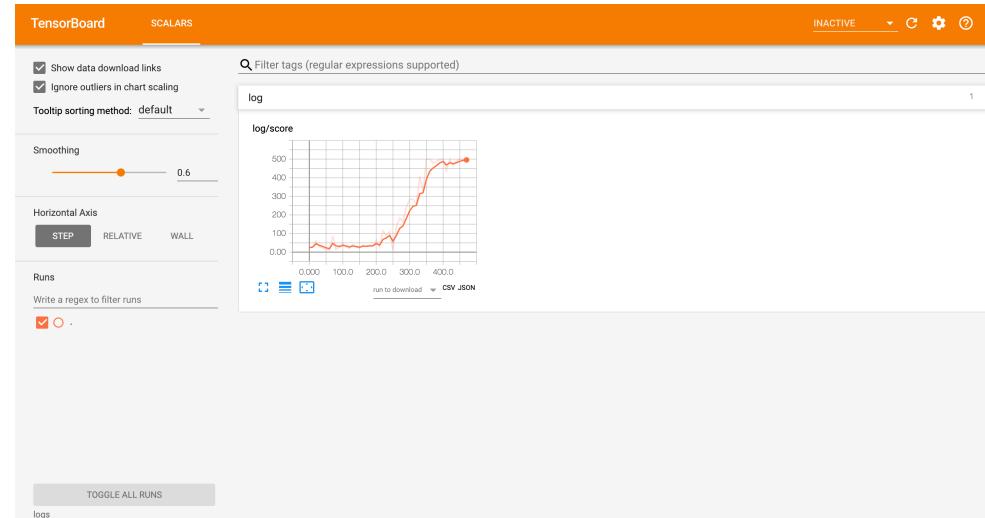
Train model

- Prediction
 - `q_values` - (32, 2)
 - `action.unsqueeze(1)` - (32, 1)
 - `q_value` - (32)
- Target
 - `next_q_values` - (32, 2)
 - `next_q_values.max(1)[1]` - (32)
 - `target_next_q_values` - (32, 2)
 - `next_q_values_index.unsqueeze(1)` - (32, 1)
 - `target` - (32)

```
45     criterion = torch.nn.MSELoss()
46
47     # get Q-value
48     q_values = q_net(torch.Tensor(states))
49     q_value = q_values.gather(1, actions.unsqueeze(1)).view(-1)
50
51     # get target
52     next_q_values = q_net(torch.Tensor(next_states))
53     next_q_value_index = next_q_values.max(1)[1]
54
55     target_next_q_values = target_q_net(torch.Tensor(next_states))
56     target_next_q_value = target_next_q_values.gather(1, next_q_value_index.unsqueeze(1)).view(-1)
57     target = rewards + masks * args.gamma * target_next_q_value
58
59     loss = criterion(q_value, target.detach())
60     optimizer.zero_grad()
61     loss.backward()
62     optimizer.step()
```

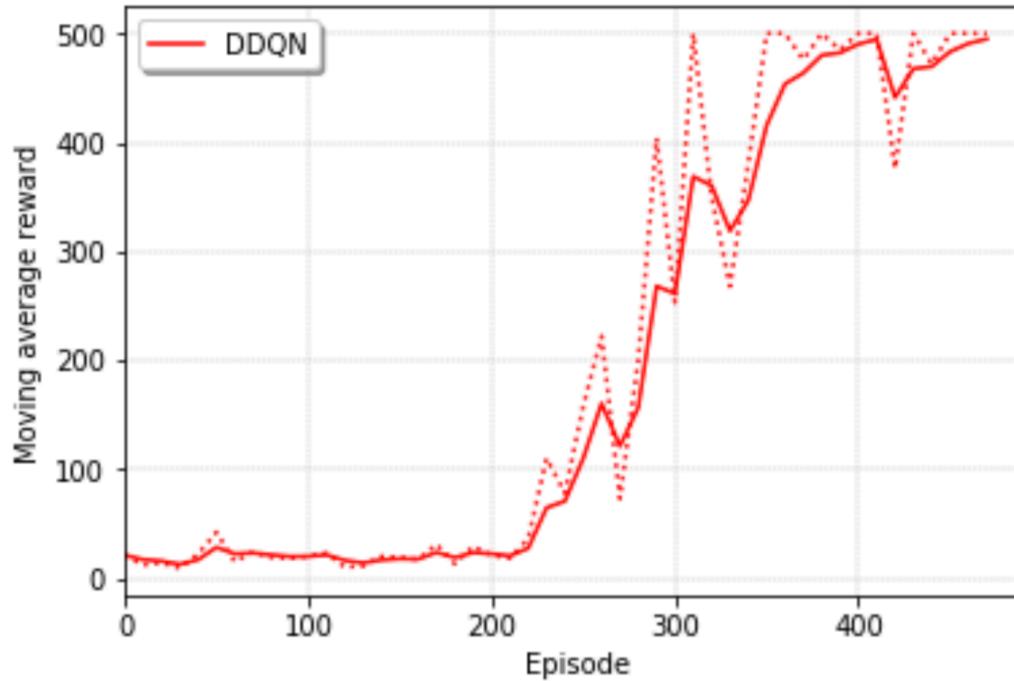
Train & TensorboardX

- Terminal A - train 실행
 - conda activate env_name
 - python train.py
- Terminal B - tensorboardX 실행
 - conda activate env_name
 - tensorboard --logdir logs
 - (웹에서) localhost:6006



Learning curve & Test

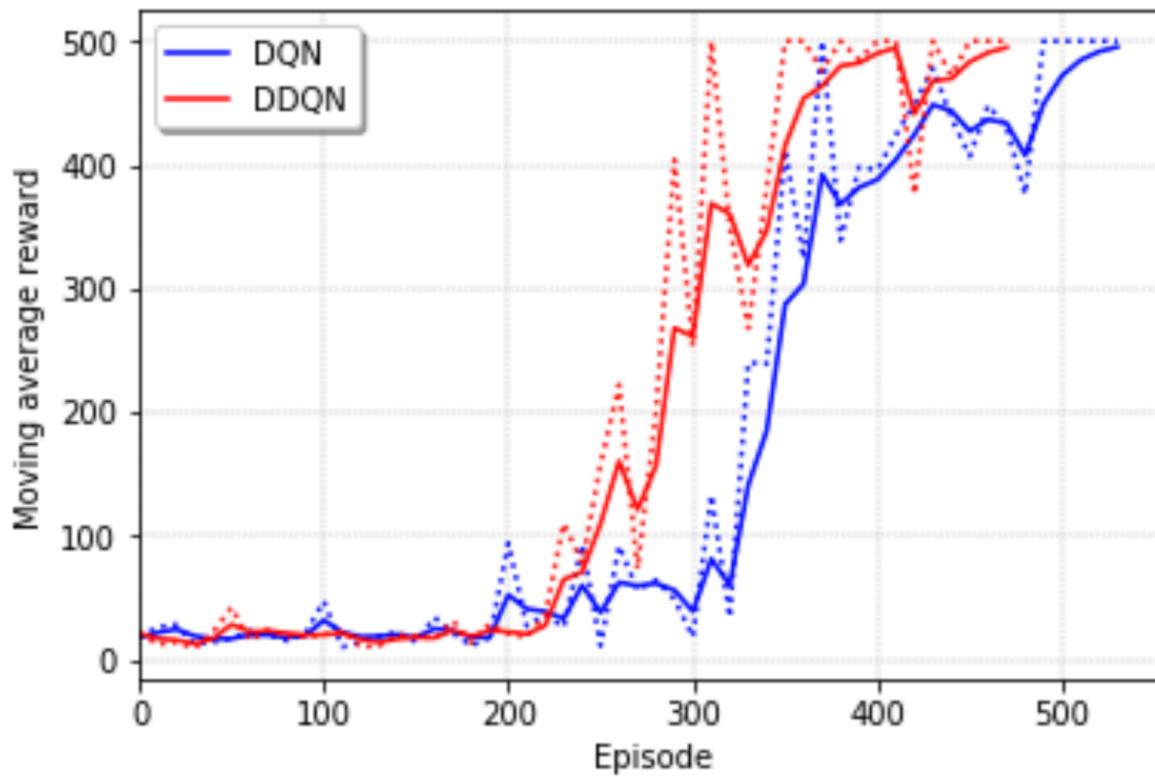
- Learning curve



- Test
 - `python test.py`

DQN vs. DDQN

- Learning curve



Thank you



CORE
Control + Optimization Research Lab