모두의연구소

Double DQN

2020.07.25

김지훈

모두의연구소

첫 논문 발표라 많이 부족합니다. 열심히 하겠습니다!!

기존의 DQN 알고리즘

특정한 조건에서 action value를 overestimation하는 문제
-> action value에 대한 maximization step이 존재한다.
-> 실제로 특정 게임에서 좋지 못한 성능을 보였다.

Overestimation

$$\hat{q}(s', a', W) = q_\pi(s', a') + Y_{s'}^{a'}$$

Q를 근사한 함수와 Q 함수의 차이는 평균이 0이 될 수 없다. Max이기 때문에.

Background

Sequential decision problem을 풀기 위해 각각의 action에 대한 optimal value에 대한 추측값을 학습할 수 있고, 이러한 action을 취하고 이에 따른 최적의정책을 따를 때 얻어지는 미래 보상들의 합의 기댓값을 정의할 수 있다.

$$Q_\pi(s,a) \equiv \mathbb{E}\left[R_1 + \gamma R_2 + \ldots \mid S_0 = s, A_0 = a, \pi\right]$$

Optimal policy는 action value 값 중에서 max 값을 선택하는 것이다.

$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

Background

기존의 DQN =>

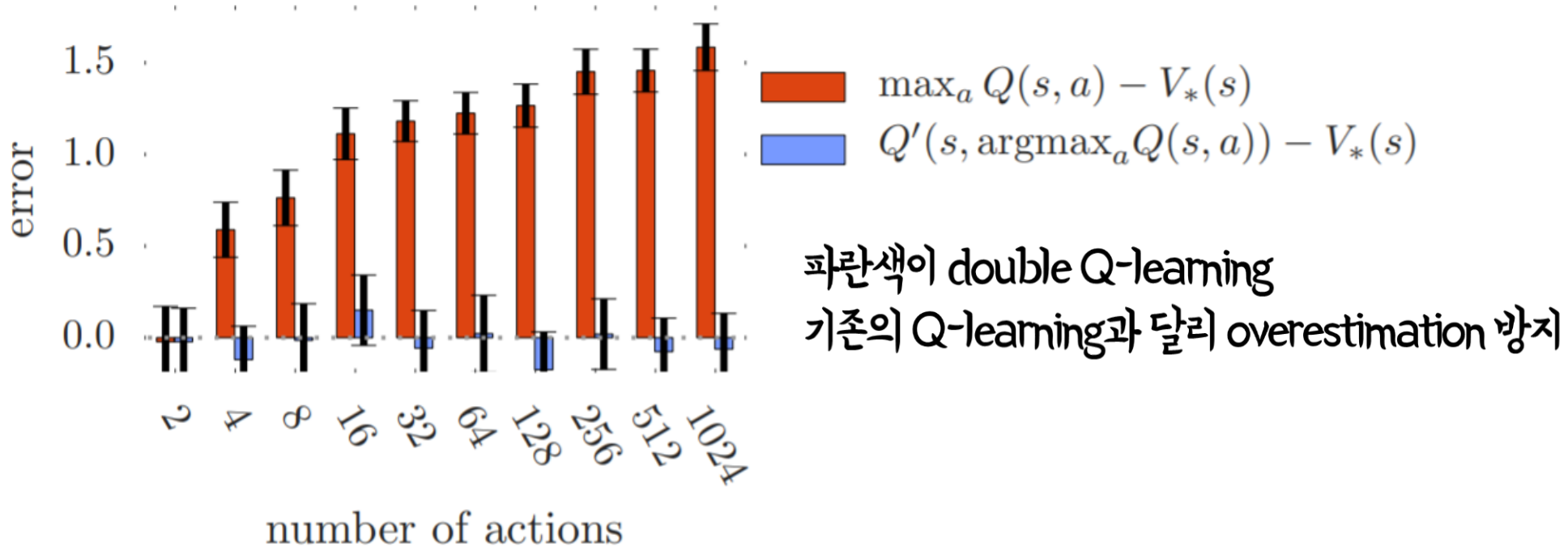
$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t)$$

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-)$$

여기의 max 연산자는 action을 select하고 evaluate할 때 같은 값을 사용한다.
기존의 Q 함수를 변형하면 $Y_t^{\mathrm{Q}} = R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t)$
Select와 evaluate를 분리하자 => Double Q-learning

$$Y_t^{\mathrm{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

Overoptimism due to estimation errors



파란색이 double Q-learning
기존의 Q-learning과 달리 overestimation 방지

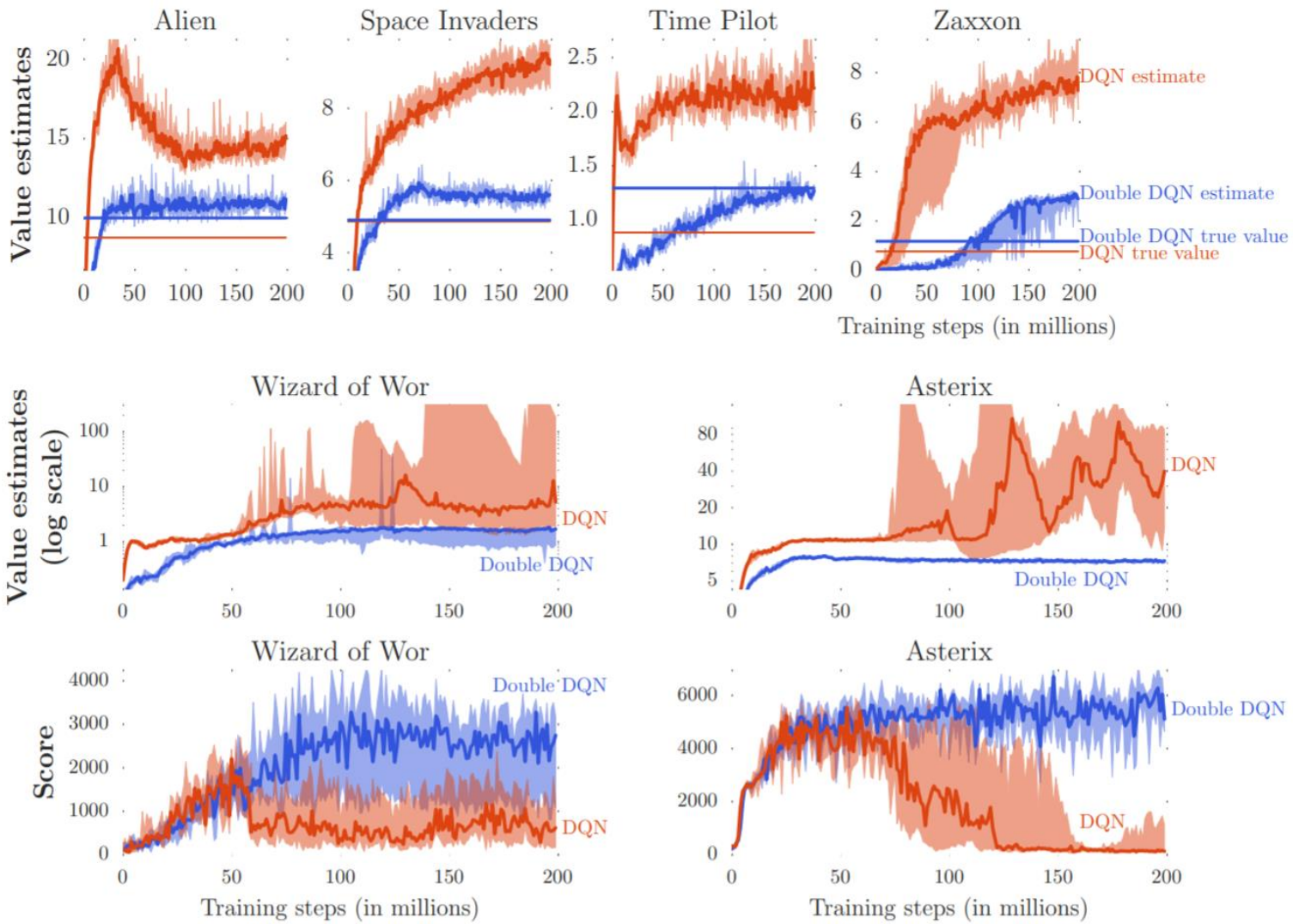
D DQN

target network

main network

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\text{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-)$$

Next state에서 Q가 최대가 되는 행동 a은 main networ에서 구하고, 그 때의 Q 값은 target network에서


Alien
Space Invaders
Time Pilot
Zaxxon
Value estimates
DQN estimate
Double DQN estimate
Double DQN true value
DQN true value
Training steps (in millions)
Wizard of Wor
Asterix
Value estimates (log scale)
DQN
Double DQN
Wizard of Wor
Asterix
Score
Double DQN
DQN
Training steps (in millions)

코드 실습

코드 : https://github.com/wikibook/pytorch-drl/blob/master/program/6_2_DDQN.ipynb

코드 실습

```
# namedtuple 생성
from collections import namedtuple

Transition = namedtuple(
    'Transition', ('state', 'action', 'next_state', 'reward'))

ENV = 'CartPole-v0'
GAMMA = 0.99
MAX_STEPS = 200
NUM_EPISODES = 500
```

코드 실습

```
class ReplayMemory:

    def __init__(self, CAPACITY):
        self.capacity = CAPACITY
        self.memory = []
        self.index = 0

    def push(self, state, action, state_next, reward):

        self.memory[self.index] = Transition(state, action, state_next, reward)

        self.index = (self.index + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

코드 실습

```
class Net(nn.Module):

    def __init__(self, n_in, n_mid, n_out):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(n_in, n_mid)
        self.fc2 = nn.Linear(n_mid, n_mid)
        self.fc3 = nn.Linear(n_mid, n_out)

    def forward(self, x):
        h1 = F.relu(self.fc1(x))
        h2 = F.relu(self.fc2(h1))
        output = self.fc3(h2)
        return output
```

코드 실습

```
BATCH_SIZE = 32
CAPACITY = 10000
```

```
class Brain:
    def __init__(self, num_states, num_actions):
        self.num_actions = num_actions
        self.memory = ReplayMemory(CAPACITY)
        n_in, n_mid, n_out = num_states, 32, num_actions
        self.main_q_network = Net(n_in, n_mid, n_out)
        self.target_q_network = Net(n_in, n_mid, n_out)

        self.optimizer = optim.Adam(
            self.main_q_network.parameters(), lr=0.0001)

    def replay(self):

        if len(self.memory) < BATCH_SIZE:
            return
        self.batch, self.state_batch, self.action_batch, self.reward_batch, self.non_final_next_states = self.make_minibatch()
        self.expected_state_action_values = self.get_expected_state_action_values()
        self.update_main_q_network()
```

코드 실습

```
def decide_action(self, state, episode):

    epsilon = 0.5 * (1 / (episode + 1))
    if epsilon <= np.random.uniform(0, 1):
        self.main_q_network.eval()
        with torch.no_grad():
            action = self.main_q_network(state).max(1)[1].view(1, 1)

    else:
        action = torch.LongTensor(
            [[random.randrange(self.num_actions)]])
    return action

def make_minibatch(self):

    transitions = self.memory.sample(BATCH_SIZE)
    batch = Transition(*zip(*transitions))
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)
    non_final_next_states = torch.cat([s for s in batch.next_state
                                       if s is not None])

    return batch, state_batch, action_batch, reward_batch, non_final_next_states
```

코드 실습

```
def get_expected_state_action_values(self):
    self.main_q_network.eval()
    self.target_q_network.eval()
    self.state_action_values = self.main_q_network(
        self.state_batch).gather(1, self.action_batch)
    non_final_mask = torch.ByteTensor(tuple(map(lambda s: s is not None,
                                                self.batch.next_state)))
    next_state_values = torch.zeros(BATCH_SIZE)
    a_m = torch.zeros(BATCH_SIZE).type(torch.LongTensor)
    a_m[non_final_mask] = self.main_q_network(
        self.non_final_next_states).detach().max(1)[1]
    a_m_non_final_next_states = a_m[non_final_mask].view(-1, 1)
    next_state_values[non_final_mask] = self.target_q_network(
        self.non_final_next_states).gather(1, a_m_non_final_next_states).detach().squeeze()
    expected_state_action_values = self.reward_batch + GAMMA * next_state_values

    return expected_state_action_values

def update_main_q_network(self):

    self.main_q_network.train()
    loss = F.smooth_l1_loss(self.state_action_values,
                            self.expected_state_action_values.unsqueeze(1))
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()

def update_target_q_network(self):
    self.target_q_network.load_state_dict(self.main_q_network.state_dict())
```

코드 실습

```
class Agent:
    def __init__(self, num_states, num_actions):
        self.brain = Brain(num_states, num_actions)

    def update_q_function(self):
        self.brain.replay()

    def get_action(self, state, episode):
        action = self.brain.decide_action(state, episode)
        return action

    def memorize(self, state, action, state_next, reward):
        self.brain.memory.push(state, action, state_next, reward)

    def update_target_q_function(self):
        self.brain.update_target_q_network()
```

코드 실습

```
class Environment:

    def __init__(self):
        self.env = gym.make(ENV)
        num_states = self.env.observation_space.shape[0]
        num_actions = self.env.action_space.n
        self.agent = Agent(num_states, num_actions)
```

```
def run(self):
    episode_10_list = np.zeros(10)
    complete_episodes = 0
    episode_final = False
```

```
for episode in range(NUM_EPISODES):
    observation = self.env.reset()

    state = observation
    state = torch.from_numpy(state).type(
        torch.FloatTensor)
    state = torch.unsqueeze(state, 0)
```

코드 실습

```
for step in range(MAX_STEPS):
    action = self.agent.get_action(state, episode)
    observation_next, _, done, _ = self.env.step(
        action.item())
    if done:
        state_next = None

        episode_10_list = np.hstack(
            (episode_10_list[1:], step + 1))

        if step < 195:
            reward = torch.FloatTensor(
                [-1.0])
            complete_episodes = 0
        else:
            reward = torch.FloatTensor([1.0])
            complete_episodes = complete_episodes + 1
    else:
        reward = torch.FloatTensor([0.0])
        state_next = observation_next
        state_next = torch.from_numpy(state_next).type(
            torch.FloatTensor)
        state_next = torch.unsqueeze(state_next, 0)
    self.agent.memorize(state, action, state_next, reward)
    self.agent.update_q_function()
    state = state_next
```

코드 실습

```
        if done:
            print('%d Episode: Finished after %d steps：최근 10 에피소드의 평균 단계 수 = %.1lf' % (
                episode, step + 1, episode_10_list.mean()))

            if(episode % 2 == 0):
                self.agent.update_target_q_function()
            break


    if episode_final is True:
        break

    if complete_episodes >= 10:
        print('10 에피소드 연속 성공')
        episode_final = True
```

코드 실습

```
cartpole_env = Environment()
cartpole_env.run()
```

DQN (autosaved)

View Insert Cell Kernel Widgets Help Trusted Python

Run Markdown

```
cartpole_env = Environment()
cartpole_env.run()
```

```
169 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 197.5
170 Episode: Finished after 181 steps : 최근 10 에피소드의 평균 단계 수 = 196.9
171 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 196.9
172 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 196.9
173 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 196.9
174 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 196.9
175 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 198.1
176 Episode: Finished after 198 steps : 최근 10 에피소드의 평균 단계 수 = 197.9
177 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 197.9
178 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 197.9
179 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 197.9
180 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 199.8
10 에피소드 연속 성공
```

```
MovieWriter ffmpeg unavailable. Trying to use pillow instead.
```

```
181 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 199.8
```

200724 DDQN (autosaved)

View Insert Cell Kernel Widgets Help Trusted Pytho

Run Code

```
118 Episode: Finished after 134 steps : 최근 10 에피소드의 평균 단계 수 = 120.7
119 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 127.3
120 Episode: Finished after 150 steps : 최근 10 에피소드의 평균 단계 수 = 133.4
121 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 142.6
122 Episode: Finished after 171 steps : 최근 10 에피소드의 평균 단계 수 = 148.5
123 Episode: Finished after 192 steps : 최근 10 에피소드의 평균 단계 수 = 157.8
124 Episode: Finished after 197 steps : 최근 10 에피소드의 평균 단계 수 = 166.4
125 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 175.2

126 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 182.7
127 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 184.4
128 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 191.0
129 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 191.0
130 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 196.0
131 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 196.0
132 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 198.9
133 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 199.7
10 에피소드 연속 성공
134 Episode: Finished after 200 steps : 최근 10 에피소드의 평균 단계 수 = 200.0
```

출처

논문 : https://arxiv.org/abs/1509.06461

코드 : https://github.com/wikibook/pytorch-drl/blob/master/program/6_2_DDQN.ipynb

참고
https://jsideas.net/dqn/

모두의연구소

감사합니다!