DQN 구현

[쉽게 구현하는 강화학습 2화]

팡요랩 – 노승은, 전민영

2019.05.12

https://github.com/seungeunrho/minimalRL

복습 – (6장)

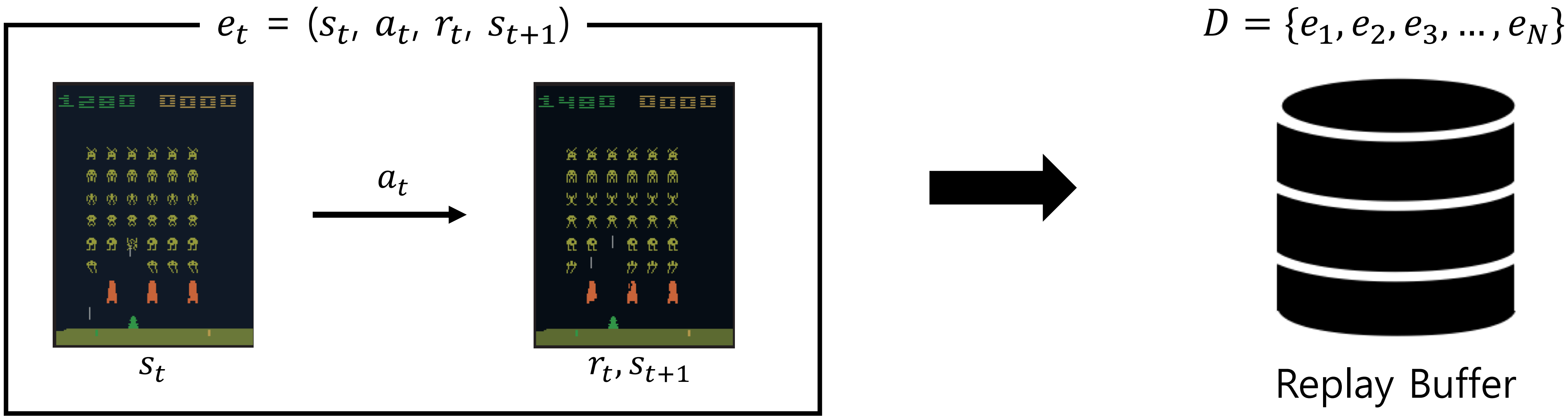
Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector $\mathbf{w}$ minimising mean-squared error between approximate value fn $\hat{v}(s, \mathbf{w})$ and true value fn $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

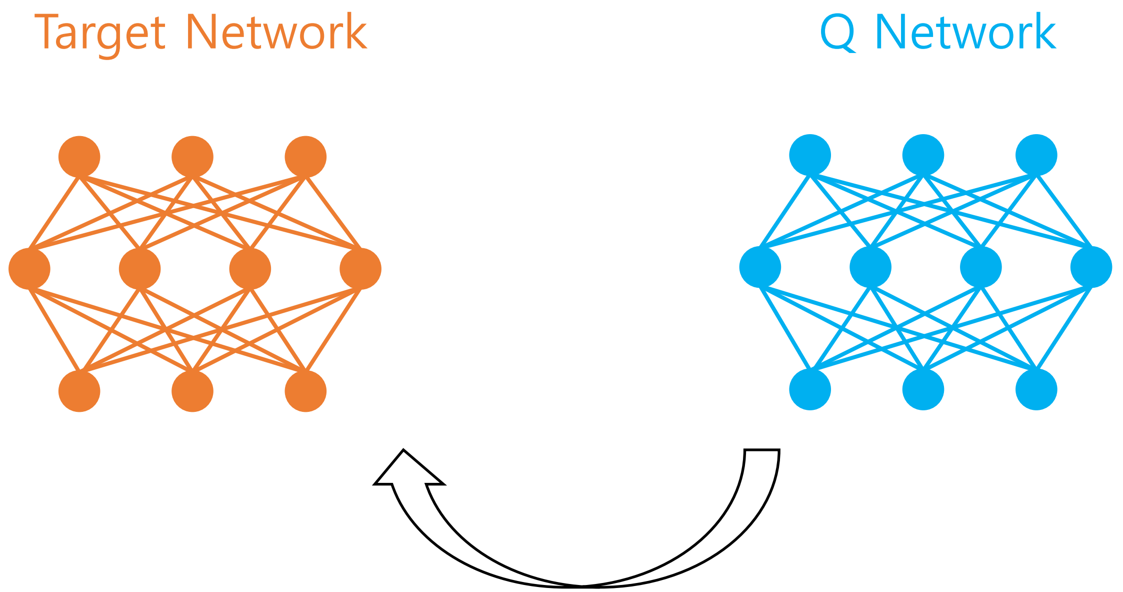
- Have assumed true value function $v_\pi(s)$ given by supervisor
- In practice, we substitute a *target* for $v_\pi(s)$
- For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

복습 – Replay Buffer



복습 – Target Network

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$



일정 주기마다 update

$$\theta_i^- = \theta_i$$

Import

```python
import gym
import collections
import random

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

- Collections library는 replay buffer에서 쓰일 deque 를 import 하기 위함.

Replay Buffer

```python
class ReplayBuffer():
    def __init__(self):
        self.buffer = collections.deque()
        self.batch_size = 32
        self.size_limit = 50000

    def put(self, data):
        self.buffer.append(data)
        if len(self.buffer) > self.size_limit:
            self.buffer.popleft()

    def sample(self, n):
        return random.sample(self.buffer, n)

    def size(self):
        return len(self.buffer)
```

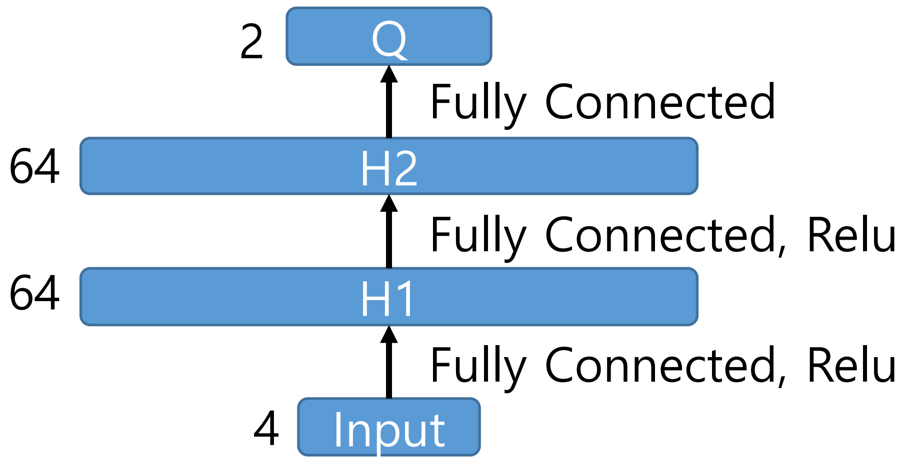
Buffer의 최대 크기

Q Network

```python
class Qnet(nn.Module):
    def __init__(self):
        super(Qnet, self).__init__()
        self.fc1 = nn.Linear(4, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 2)

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

    def sample_action(self, obs, epsilon):
        out = self.forward(obs)
        coin = random.random()
        if coin < epsilon:
            return random.randint(0,1)
        else :
            return out.argmax().item()
```



Main 1

```python
def main():
    env = gym.make('CartPole-v1')
    q = Qnet()
    q_target = Qnet()
    q_target.load_state_dict(q.state_dict())
    memory = ReplayBuffer()

    avg_t = 0
    gamma = 0.98
    batch_size = 32
    optimizer = optim.Adam(q.parameters(), lr=0.0005)
```

- Q와 Target Q 네트워크 두 개를 선언

- Q를 Target Q로 복사
- state_dict 는 model의 weight 정보를 dictionary형태로 담고 있음
- 예컨대 아래와 같음

```
Model's state_dict:
conv1.weight     torch.Size([6, 3, 5, 5])
conv1.bias     torch.Size([6])
conv2.weight     torch.Size([16, 6, 5, 5])
conv2.bias     torch.Size([16])
fc1.weight     torch.Size([120, 400])
fc1.bias     torch.Size([120])
fc2.weight     torch.Size([84, 120])
fc2.bias     torch.Size([84])
fc3.weight     torch.Size([10, 84])
fc3.bias     torch.Size([10])
```

Main 2

```python
for n_epi in range(10000):
    epsilon = max(0.01, 0.08 - 0.01*(n_epi/200)) #Linear annealing from 8% to 1%
    s = env.reset()

    for t in range(600):
        a = q.sample_action(torch.from_numpy(s).float(), epsilon)
        s_prime, r, done, info = env.step(a)
        done_mask = 0.0 if done else 1.0
        memory.put((s,a,r/200.0,s_prime, done_mask))
        s = s_prime

        if done:
            break
```

```python
if memory.size()>2000:
    train(q, q_target, memory, gamma, optimizer, batch_size)

if n_epi%20==0 and n_epi!=0:
    q_target.load_state_dict(q.state_dict())
    print("# of episode :{}, Avg timestep : {:.1f}, buffer size : {}, epsilon : {:.1f}%".format(
                                                        n_epi, avg_t/20.0, memory.size(), epsilon*100))
```

```python
def train(q, q_target, memory, gamma, optimizer, batch_size):
    for i in range(10):
        batch = memory.sample(batch_size)
        s_lst, a_lst, r_lst, s_prime_lst, done_mask_lst = [], [], [], [], []

        for transition in batch:
            s, a, r, s_prime, done_mask = transition
            s_lst.append(s)
            a_lst.append([a])
            r_lst.append([r])
            s_prime_lst.append(s_prime)
            done_mask_lst.append([done_mask])

        s,a,r,s_prime,done_mask = torch.tensor(s_lst, dtype=torch.float), torch.tensor(a_lst), \
                                  torch.tensor(r_lst), torch.tensor(s_prime_lst, dtype=torch.float), \
                                  torch.tensor(done_mask_lst)
        q_out = q(s)
        q_a = q_out.gather(1,a)
        max_q_prime = q_target(s_prime).max(1)[0].unsqueeze(1)
        target = r + gamma * max_q_prime * done_mask
        loss = F.smooth_l1_loss(target, q_a)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Shape : [32,2]

취한 action의 q값만 골라냄.
Shape : [32,1]

$$r+\gamma \max_{a'} Q(s',a';\theta_i^-)$$

학습 결과

```
# of episode :20, Avg timestep : 9.1, buffer size : 202, epsilon : 7.9%
# of episode :40, Avg timestep : 8.6, buffer size : 393, epsilon : 7.8%
# of episode :60, Avg timestep : 8.7, buffer size : 587, epsilon : 7.7%
# of episode :80, Avg timestep : 8.6, buffer size : 779, epsilon : 7.6%
# of episode :100, Avg timestep : 8.7, buffer size : 973, epsilon : 7.5%
# of episode :120, Avg timestep : 8.6, buffer size : 1165, epsilon : 7.4%
# of episode :140, Avg timestep : 8.8, buffer size : 1360, epsilon : 7.3%
# of episode :160, Avg timestep : 9.0, buffer size : 1560, epsilon : 7.2%
# of episode :180, Avg timestep : 8.7, buffer size : 1754, epsilon : 7.1%
# of episode :200, Avg timestep : 8.7, buffer size : 1947, epsilon : 7.0%
# of episode :220, Avg timestep : 10.6, buffer size : 2179, epsilon : 6.9%
# of episode :240, Avg timestep : 14.6, buffer size : 2491, epsilon : 6.8%
# of episode :260, Avg timestep : 10.9, buffer size : 2729, epsilon : 6.7%
# of episode :280, Avg timestep : 9.9, buffer size : 2947, epsilon : 6.6%
# of episode :300, Avg timestep : 17.4, buffer size : 3316, epsilon : 6.5%
# of episode :320, Avg timestep : 109.5, buffer size : 5525, epsilon : 6.4%
# of episode :340, Avg timestep : 121.2, buffer size : 7970, epsilon : 6.3%
# of episode :360, Avg timestep : 213.4, buffer size : 12259, epsilon : 6.2%
# of episode :380, Avg timestep : 179.5, buffer size : 15869, epsilon : 6.1%
# of episode :400, Avg timestep : 114.1, buffer size : 18171, epsilon : 6.0%
# of episode :420, Avg timestep : 101.2, buffer size : 20215, epsilon : 5.9%
# of episode :440, Avg timestep : 139.5, buffer size : 23025, epsilon : 5.8%
# of episode :460, Avg timestep : 163.5, buffer size : 26315, epsilon : 5.7%
# of episode :480, Avg timestep : 199.3, buffer size : 30322, epsilon : 5.6%
# of episode :500, Avg timestep : 268.6, buffer size : 35715, epsilon : 5.5%
# of episode :520, Avg timestep : 260.0, buffer size : 40935, epsilon : 5.4%
# of episode :540, Avg timestep : 207.1, buffer size : 45096, epsilon : 5.3%
# of episode :560, Avg timestep : 234.8, buffer size : 49812, epsilon : 5.2%
# of episode :580, Avg timestep : 209.8, buffer size : 50000, epsilon : 5.1%
```