

PPO 구현

[쉽게 구현하는 강화학습 3화]

판요랩 - 노승은, 전민영

2019.05.26

<https://github.com/seungeunrho/minimalRL>

Main 1

```
83 def main():
84     env = gym.make('CartPole-v1')
85     model = PPO()
86     gamma = 0.99
87     T = 20 ----->
88     score = 0.0
89     print_interval = 20
```

- 몇 time step 동안 data를 모을지!
- 아래의 pseudo-code 에서 T와 같음.

for iteration=1, 2, ... **do**

Run policy for T timesteps or N trajectories

Estimate advantage function at all timesteps

Do SGD on $L^{CLIP}(\theta)$ objective for some number of epochs

end for

Main 2

```
for n_epi in range(10000):
    s = env.reset()
    done = False
    while not done:
        for t in range(T):
            prob = model.pi(torch.from_numpy(s).float())
            m = Categorical(prob)
            a = m.sample().item()
            s_prime, r, done, info = env.step(a)
            model.put_data((s, a, r/100.0, s_prime, prob[a].item(), done))
            s = s_prime

            score += r
            if done:
                break

        model.train()
```

실제로 T 스텝만큼 data를 모으고 학습을 함

보통은 (s, a, r, s', done) 만 저장
근데 이상한 친구가 하나 있네요?
얘는 실제 한 action의 확률값
나중에 ratio 계산할 때 쓰임

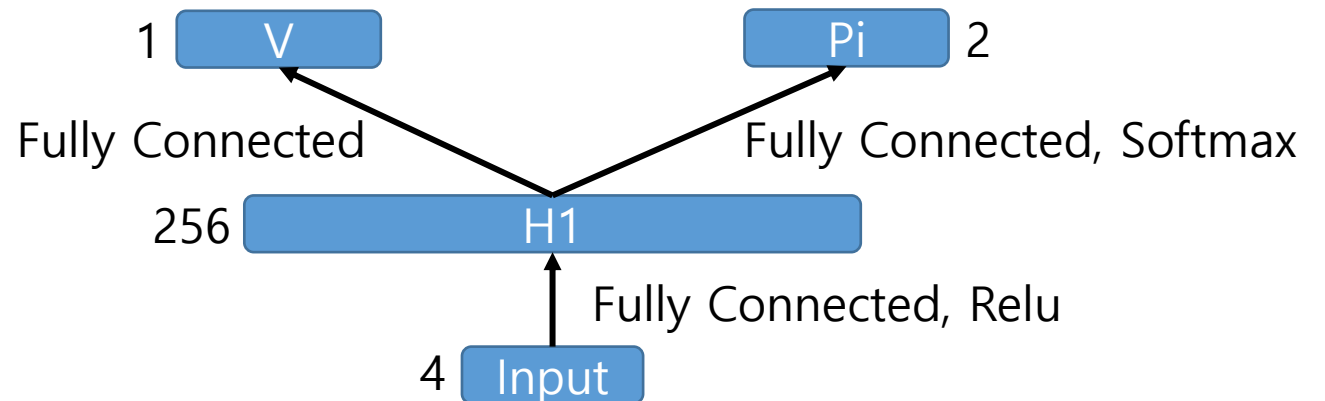
PPO - Initializer

```
8  class PPO(nn.Module):
9      def __init__(self):
10         super(PPO, self).__init__()
11         self.data = []
12         self.gamma = 0.98
13         self.lmbda = 0.95
14         self.eps = 0.1
15         self.K = 2 #epoch } 새로 생긴 친구들. 곧 나올 것임
16
17         self.fc1 = nn.Linear(4,256)
18         self.fc_pi = nn.Linear(256,2)
19         self.fc_v = nn.Linear(256,1)
20         self.optimizer = optim.Adam(self.parameters(), lr=0.0005)
```

PPO - Policy, Value Network

```
22     def pi(self, x, softmax_dim = 0):
23         x = F.relu(self.fc1(x))
24         x = self.fc_pi(x)
25         prob = F.softmax(x, dim=softmax_dim)
26         return prob
27
28     def v(self, x):
29         x = F.relu(self.fc1(x))
30         v = self.fc_v(x)
31         return v
```

- Batch 처리를 위해 도입 된 친구.
- 그냥 s_t 에 대해 inference를 할 때는 $\text{dim} = 0$
- 하지만 $[s_1, s_2, \dots, s_t]$ 가 input으로 들어간다면?
- Softmax $\text{dim} = 1$ 이어야 함
- 그래서 pi 함수에 인자로 넘겨주는 식으로 구현.



PPO - Data Processing

```
def put_data(self, item):  
    self.data.append(item)
```

```
def make_batch(self):  
    s_lst, a_lst, r_lst, s_prime_lst, prob_a_lst, done_lst = [], [], [], [], [], []  
    for item in self.data:  
        s, a, r, s_prime, prob_a, done = item  
  
        s_lst.append(s)  
        a_lst.append([a])  
        r_lst.append([r])  
        s_prime_lst.append(s_prime)  
        prob_a_lst.append([prob_a])  
        done_mask = 0 if done else 1  
        done_lst.append([done_mask])
```

- S는 numpy array임.
- 그 shape은 [0.3, 0.6, -0.2, 1.6] 이런 식으로 생김.
- 따라서 다른 a, r, done_mask, prob_a 같은 친구들도 dimension을 맞춰주기 위해 괄호 안에 넣어서 append를 해 줌.
- 안 그러면 나중에 shape이 안 맞는다고 에러가 남.

```
s,a,r,s_prime,done_mask, prob_a = 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_lst, dtype=torch.float), torch.tensor(prob_a_ls)  
  
self.data = []  
return s,a,r,s_prime,done_mask, prob_a
```

PPO - GAE (Generalized Advantage Estimation)

보통의 Advantage : $\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$

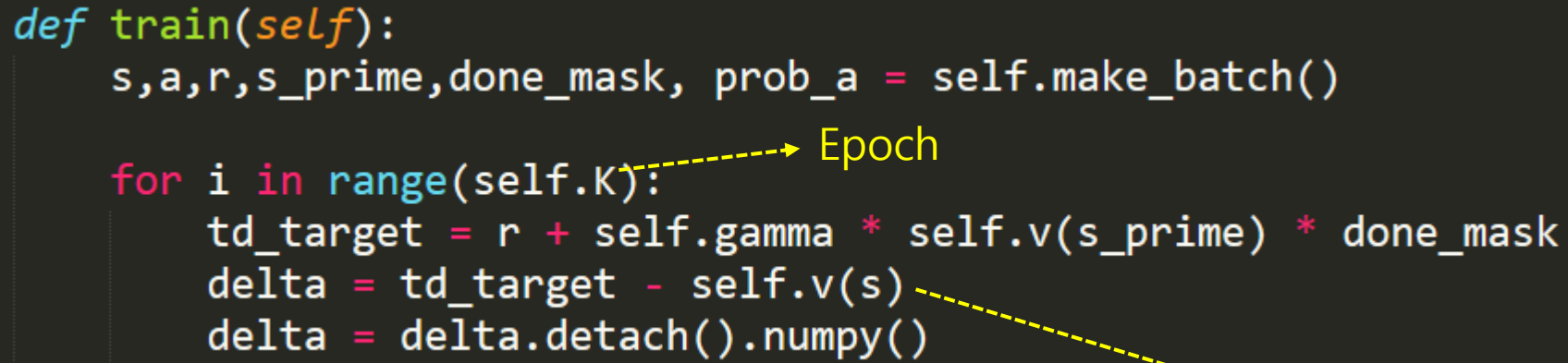
GAE : $\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$,
where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

이걸 어떻게 배치로 한방에 처리하지..?

$$\begin{aligned} \delta_1 &= r_1 + \gamma V(s_2) - V(s_1) \\ \delta_2 &= r_2 + \gamma V(s_3) - V(s_2) \\ \delta_3 &= r_3 + \gamma V(s_4) - V(s_3) \\ &\vdots \\ \delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \quad \Rightarrow \quad \begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \vdots \\ \delta_t \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_t \end{pmatrix} + \gamma \begin{pmatrix} V(s_2) \\ V(s_3) \\ V(s_4) \\ \vdots \\ V(s_{t+1}) \end{pmatrix} - \begin{pmatrix} V(s_1) \\ V(s_2) \\ V(s_3) \\ \vdots \\ V(s_t) \end{pmatrix}$$

PPO - GAE

```
def train(self):  
    s,a,r,s_prime,done_mask, prob_a = self.make_batch()  
  
    for i in range(self.K):  
        td_target = r + self.gamma * self.v(s_prime) * done_mask  
        delta = td_target - self.v(s)  
        delta = delta.detach().numpy()
```



배치로 한방에 계산됨!

$$\begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \vdots \\ \delta_t \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_t \end{pmatrix} + \gamma \begin{pmatrix} V(s_2) \\ V(s_3) \\ V(s_4) \\ \vdots \\ V(s_{t+1}) \end{pmatrix} - \begin{pmatrix} V(s_1) \\ V(s_2) \\ V(s_3) \\ \vdots \\ V(s_t) \end{pmatrix}$$

PPO - GAE

$$\begin{pmatrix} \delta_t \\ \delta_{t+1} \\ \delta_{t+2} \\ \vdots \\ \delta_{T-1} \end{pmatrix}$$

```
advantage_lst = []
advantage = 0
for delta_t in delta[::-1]:
    advantage = self.gamma * self.lmbda * advantage + delta_t[0]
    advantage_lst.append([advantage])
advantage_lst.reverse()
advantage = torch.tensor(advantage_lst, dtype=torch.float)
```

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$

$$\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

- 델타를 뒤에서 부터 보면서 γ 랑 λ 를 곱하고
- 계속해서 더해나가고
- 중간중간 이거를 리스트에 저장해두면
- 그게 바로 advantage! (의 거꾸로 쌓인 버전)
- 다시 reverse를 해줘야 함

PPO - Clipped Loss

```
pi = self.pi(s, softmax_dim=1)
pi_a = pi.gather(1, a)
ratio = torch.exp(torch.log(pi_a) - torch.log(prob_a)) # a/b == exp(log(a)-log(b))

surr1 = ratio * advantage
surr2 = torch.clamp(ratio, 1-self.eps, 1+self.eps) * advantage
loss = -torch.min(surr1, surr2) + F.smooth_l1_loss(td_target.detach(), self.v(s))

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

$$r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t) \right]$$

- 요 친구가 사실 굉장히 중요함
- Td_target이 만들어지기 까지 필요했던 앞의 그래프들을 다 떼어버리겠다는 뜻.
- 즉 gradient 의 flow가 발생하지 않음!
- 떼어내지 않으면 target 도 loss를 줄이는 방향으로 update가 됨.
- 즉, v(s')함수의 그래프까지 gradient가 전달된다는 뜻.

실험 1 - detach()를 안하면..?

```
surr1 = ratio * advantage
surr2 = torch.clamp(ratio, 1-self.eps, 1+self.eps) * advantage
loss = -torch.min(surr1, surr2) + F.smooth_l1_loss(td_target, self.v(s))
```

td_target.detach() 를 td_target 으로 바꿔봤습니다.

```
# of episode :0, avg score : 0.6
# of episode :20, avg score : 21.1
# of episode :40, avg score : 25.1
# of episode :60, avg score : 18.1
# of episode :80, avg score : 18.6
# of episode :100, avg score : 24.6
# of episode :120, avg score : 33.4
# of episode :140, avg score : 36.1
# of episode :160, avg score : 30.4
# of episode :180, avg score : 54.1
# of episode :200, avg score : 52.5
# of episode :220, avg score : 33.8
# of episode :240, avg score : 25.5
# of episode :260, avg score : 23.8
# of episode :280, avg score : 29.6
```

```
# of episode :300, avg score : 39.0
# of episode :320, avg score : 50.9
# of episode :340, avg score : 83.9
# of episode :360, avg score : 86.0
# of episode :380, avg score : 123.2
# of episode :400, avg score : 166.9
# of episode :420, avg score : 129.3
# of episode :440, avg score : 38.0
# of episode :460, avg score : 96.2
# of episode :480, avg score : 65.7
# of episode :500, avg score : 72.7
# of episode :520, avg score : 69.6
```

- 굉장히 학습이 불안정함!
- 520 episode 인데 아직 70점 언저리

Ratio 가 clipping 을 넘어가면 어떤일이 생기나?

```
pi = self.pi(s, softmax_dim=1)
pi_a = pi.gather(1, a)
ratio = torch.exp(torch.log(pi_a) - torch.log(prob_a)) # a/b == exp(log(a)-log(b))

surr1 = ratio * advantage
surr2 = torch.clamp(ratio, 1-self.eps, 1+self.eps) * advantage
loss = -torch.min(surr1, surr2) + F.smooth_l1_loss(td_target, self.v(s))
```

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t) \right]$$

- ratio == 1.3 인 상황, advantage 는 양수인 상황을 생각해보자
- eps == 0.1 인경우, Clip(ratio, 1.1, 0.9) == 1.1 이다.
- 그러면 surr1 은 surr2 보다 크니까 surr1 은 loss 에 영향을 못 미침
- surr2 에서도 ratio는 clip 값에 영향을 못미침, 즉 loss 에 영향을 못 미침
- 즉, clip 범위를 벗어나면 ratio 값이 바뀌어도 loss 값이 안 바뀐다.
(예컨대 ratio가 1.3에서 1.28이 되어도 loss는 안 변함)
- 범위를 벗어나면 그 sample은 버리는 효과임!

제대로 된 학습 결과

☞ # of episode :0, avg score : 2.3
of episode :20, avg score : 20.1
of episode :40, avg score : 28.6
of episode :60, avg score : 35.2
of episode :80, avg score : 49.8
of episode :100, avg score : 109.8
of episode :120, avg score : 129.1
of episode :140, avg score : 106.5
of episode :160, avg score : 184.0
of episode :180, avg score : 193.8
of episode :200, avg score : 241.6
of episode :220, avg score : 285.5
of episode :240, avg score : 257.5
of episode :260, avg score : 249.4
of episode :280, avg score : 248.1
of episode :300, avg score : 449.4
of episode :320, avg score : 230.6
of episode :340, avg score : 408.1
of episode :360, avg score : 475.3
of episode :380, avg score : 500.0

- 굉장히 학습이 잘됨!
- DQN, REINFORCE 보다 훨씬 좋은 느낌
- 20개 episode의 평균 점수 500점을 380 episode 만에 찍음
- (물론 학습 결과는 variance가 커서 할 때마다 다를 수 있지만 그걸 고려해도 잘 되는 느낌적 느낌)

END