Policy Gradient 구현

reinforce, TD Actor-Critic

[쉽게 구현하는 강화학습 1화]

팡요랩 – 노승은, 전민영

2019.04.27

https://github.com/seungeunrho/minimalRL

복습 – Policy Gradient (7장)

- The policy gradient has many equivalent forms

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right] && \text{TD}(\lambda)\text{ Actor-Critic} \\
G_\theta^{-1}\nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}
$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$ or $V^\pi(s)$

복습 – Policy Gradient (7장)

- For the true value function $V^{\pi_\theta}(s)$, the TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

- is an unbiased estimate of the advantage function

$$\begin{aligned} \mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta}|s,a\right] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V^{\pi_\theta}(s')|s,a\right] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s,a) \end{aligned}$$

- So we can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta^{\pi_\theta}\right]$$

- In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

AutoDiff 이용하기

구해야 하는 것 : $\nabla_\theta \log \pi_\theta(s, a) r \; = \; r \; \times \; \nabla_\theta \log \pi_\theta(s, a)$

누구를 미분하면 위에 식이 되지? $\quad r \; \times \; \log \pi_\theta(s, a)$

그러면 loss 는 어떻게? $\quad -r \; \times \; \log \pi_\theta(s, a)$

- 는 왜붙지 ? Loss 는 자동으로 minimize 되는데 우리는 maximize 하고 싶으니까!

REINFORCE

```python
#REINFORCE
import gym
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical
```

Import 부터!

REINFORCE

```python
def main():
    env = gym.make('CartPole-v1')
    pi = Policy()
    avg_t = 0

    for n_epi in range(10000):
        obs = env.reset()
        for t in range(600):
            obs = torch.tensor(obs, dtype=torch.float)
            out = pi(obs)
            m = Categorical(out)
            action = m.sample()
            obs, r, done, info = env.step(action.item())
            pi.put_data((r,torch.log(out[action])))
            if done:
                break
        avg_t += t
        pi.train()
        if n_epi%20==0 and n_epi!=0:
            print("# of episode :{}, Avg timestep : {}".format(n_epi, avg_t/20.0))
            avg_t = 0
    env.close()
```

tensor([0.0255, -0.0159, -0.0489, -0.0408])

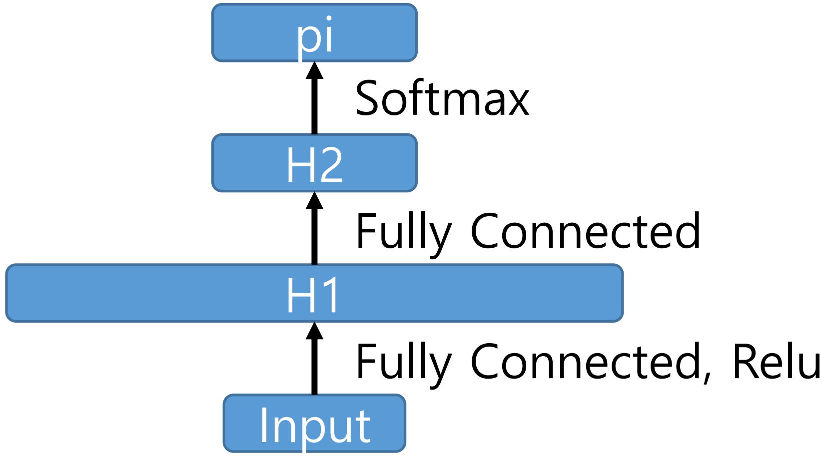
tensor(0)

$\log \pi_\theta(s, a)$

REINFORCE

```python
class Policy(nn.Module):
    def __init__(self):
        super(Policy, self).__init__()
        self.data = []
        self.gamma = 0.99

        self.fc1 = nn.Linear(4, 128)
        self.fc2 = nn.Linear(128, 2)
        self.optimizer = optim.Adam(self.parameters(), lr=0.0005)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.softmax(self.fc2(x), dim=0)
        return x

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

    def train(self):
        R = 0
        for r, log_prob in self.data[::-1]:
            R = r + R * self.gamma
            loss = -log_prob * R
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()
        self.data = []

```

$\log \pi_\theta(s, a) \; v_t$



TD Actor Critic

```python
for n_epi in range(10000):
    obs = env.reset()
    loss_lst = []
    for t in range(600):
        obs = torch.from_numpy(obs).float()
        pi, v = model(obs)
        m = Categorical(pi)
        action = m.sample()

        obs, r, done, info = env.step(action.item())
        _, next_v = model(torch.from_numpy(obs).float())
        delta = r + gamma * next_v - v
        loss = -torch.log(pi[action]) * delta.item() + delta * delta
        model.gather_loss(loss)

        if done:
            break

    model.train()
```

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

$\log \pi_\theta(s, a)\ \delta^{\pi_\theta}$

Value loss

TD Actor Critic

```python
class ActorCritic(nn.Module):
    def __init__(self):
        super(ActorCritic, self).__init__()
        self.loss_lst = []

        self.fc1 = nn.Linear(4, 128)
        self.fc_pi = nn.Linear(128, 2)
        self.fc_v = nn.Linear(128, 1)
        self.optimizer = optim.Adam(self.parameters(), lr=0.002)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        pol = self.fc_pi(x)
        pi = F.softmax(pol, dim=0)
        v = self.fc_v(x)
        return pi, v

    def gather_loss(self, loss):
        self.loss_lst.append(loss.unsqueeze(0))

    def train(self):
        loss = torch.cat(self.loss_lst).sum()
        loss = loss/len(self.loss_lst)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        self.loss_lst = []

```