

Proximal Policy Optimization (PPO)

이동민

삼성전자 서울대 공동연구소
Jul 18, 2019

Outline

- Proximal Policy Optimization (PPO)
 - Learning process
 - Hyperparameter
 - Main loop
 - Train model
 - Train & TensorboardX
 - Learning curve & Test
- TRPO vs. PPO - Learning curve

Proximal Policy Optimization (PPO)

- PPO Algorithm

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_k}(a_t | s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

Proximal Policy Optimization (PPO)

- Actor network

```
4  class Actor(nn.Module):
5      def __init__(self, state_size, action_size, args):
6          super(Actor, self).__init__()
7          self.fc1 = nn.Linear(state_size, args.hidden_size)
8          self.fc2 = nn.Linear(args.hidden_size, args.hidden_size)
9          self.fc3 = nn.Linear(args.hidden_size, action_size)
10
11     def forward(self, x):
12         x = torch.tanh(self.fc1(x))
13         x = torch.tanh(self.fc2(x))
14
15         mu = self.fc3(x)
16         log_std = torch.zeros_like(mu)
17         std = torch.exp(log_std)
18
19         return mu, std
```

Proximal Policy Optimization (PPO)

- Critic network

```
21  class Critic(nn.Module):  
22      def __init__(self, state_size, args):  
23          super(Critic, self).__init__()  
24          self.fc1 = nn.Linear(state_size, args.hidden_size)  
25          self.fc2 = nn.Linear(args.hidden_size, args.hidden_size)  
26          self.fc3 = nn.Linear(args.hidden_size, 1)  
27  
28      def forward(self, x):  
29          x = torch.tanh(self.fc1(x))  
30          x = torch.tanh(self.fc2(x))  
31          value = self.fc3(x)  
32  
33      return value  
34
```

Proximal Policy Optimization (PPO)

- 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) 을 trajectories set에 저장
 4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
 - Step 1 : Return 구하기
 - Step 2 : Mini batch 형태로 Actor & Critic network 업데이트

Learning process

1. 상태에 따른 행동 선택

```
136 | mu, std = actor(torch.Tensor(state))  
137 | action = get_action(mu, std)
```

```
def get_action(mu, std):  
    normal = Normal(mu, std)  
    action = normal.sample()  
  
    return action.data.numpy()
```

- `Normal(mu, std)` - std를 1로 고정함으로써 일정한 폭을 가지는
normal distribution에서 sampling



CORE
Control + Optimization Research Lab

Learning process

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

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

3. Sample (s, a, r) 을 trajectories set에 저장

```
119 | | | | trajectories = deque()  
141 | | | | mask = 0 if done else 1  
142 | | | |  
143 | | | | trajectories.append((state, action, reward, mask))
```

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트

- Step 1 : Return 구하기

```
def get_returns(rewards, masks, gamma):
    returns = torch.zeros_like(rewards)
    running_returns = 0

    for t in reversed(range(0, len(rewards))):
        running_returns = rewards[t] + masks[t] * gamma * running_returns
        returns[t] = running_returns

    return returns
```

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트

- Step 2 : Mini batch 형태로 Actor & Critic network 업데이트
 - Mini batch로 나누기

```
52     criterion = torch.nn.MSELoss()
53
54     n = len(states)
55     arr = np.arange(n)
56
57     for _ in range(args.model_update_num):
58         np.random.shuffle(arr)
59
60         for i in range(n // args.batch_size):
61             mini_batch_index = arr[args.batch_size * i : args.batch_size * (i + 1)]
62             mini_batch_index = torch.LongTensor(mini_batch_index)
63
64             states_samples = torch.Tensor(states)[mini_batch_index]
65             actions_samples = torch.Tensor(actions)[mini_batch_index]
```

args.batch_size * i 0
args.batch_size * (i + 1) 64

args.batch_size * i 64
args.batch_size * (i + 1) 128

args.batch_size * i 128
args.batch_size * (i + 1) 192

.

.

args.batch_size * i 1984
args.batch_size * (i + 1) 2048

args.batch_size * i 2048
args.batch_size * (i + 1) 2112

args.batch_size * i 2112
args.batch_size * (i + 1) 2176



CORE
Control + Optimization Research Lab

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트

- Step 2 : Mini batch 형태로 Actor & Critic network 업데이트
 - Critic Loss

$$J_V(\phi) = \frac{(V_\phi(s) - R)^2}{\text{Prediction} \quad \text{Target}}$$

```
67 |         # get critic loss
68 |         values_samples = critic(states_samples)
69 |         targets_samples = returns.unsqueeze(1)[mini_batch_index]
70 |
71 |         critic_loss = criterion(values_samples, targets_samples)
```

Learning process

- The probability ratio $r_t(\theta)$

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

- Surrogate objective function of TRPO

$$\mathcal{L}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A}_t \right] = [r_t(\theta) \hat{A}_t]$$

- Main objective function of PPO

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

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트

- Step 2 : Mini batch 형태로 Actor & Critic network 업데이트

- Actor Loss (clip param : 0.2)

$$J_{\pi}(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)$$

```
73     # get actor loss
74     actor_loss, ratio, advantages_samples = surrogate_loss(actor, values_samples, targets_samples,
75     states_samples, old_policy.detach(),
76     actions_samples, mini_batch_index)
77
78     clipped_ratio = torch.clamp(ratio,
79             1.0 - args.clip_param,
80             1.0 + args.clip_param)
81     clipped_actor_loss = clipped_ratio * advantages_samples
82
83     actor_loss = -torch.min(actor_loss, clipped_actor_loss).mean()
```

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트

- Step 2 : Mini batch 형태로 Actor & Critic network 업데이트
 - Actor Loss (clip param : 0.2)

$$J_{\pi}(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)$$

```
def surrogate_loss(actor, values, targets, states, old_policy, actions, batch_index):  
    mu, std = actor(torch.Tensor(states))  
    new_policy = get_log_prob(actions, mu, std)  
  
    old_policy = old_policy[batch_index]  
    ratio = torch.exp(new_policy - old_policy)  
  
    advantages = targets - values  
  
    surrogate_loss = ratio * advantages  
  
    return surrogate_loss, ratio, advantages
```

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
 - Step 2 : Mini batch 형태로 Actor & Critic network 업데이트
 - Update actor & critic

```
85          # update actor & critic
86          loss = actor_loss + 0.5 * critic_loss
87
88          critic_optimizer.zero_grad()
89          loss.backward(retain_graph=True)
90          critic_optimizer.step()
91
92          actor_optimizer.zero_grad()
93          loss.backward()
94          actor_optimizer.step()
```

Hyperparameter

```
14 parser = argparse.ArgumentParser()
15 parser.add_argument('--env_name', type=str, default="Pendulum-v0")
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('--lamda', type=float, default=0.98)
21 parser.add_argument('--hidden_size', type=int, default=64)
22 parser.add_argument('--batch_size', type=int, default=64)
23 parser.add_argument('--actor_lr', type=float, default=1e-3)
24 parser.add_argument('--critic_lr', type=float, default=1e-3)
25 parser.add_argument('--model_update_num', type=int, default=10)
26 parser.add_argument('--clip_param', type=float, default=0.2)
27 parser.add_argument('--max_iter_num', type=int, default=500)
28 parser.add_argument('--total_sample_size', type=int, default=2048)
29 parser.add_argument('--log_interval', type=int, default=5)
30 parser.add_argument('--goal_score', type=int, default=-300)
31 parser.add_argument('--logdir', type=str, default='./logs',
32 |   |   |   |   |   help='tensorboardx logs directory')
33 args = parser.parse_args()
```



Main loop

- Initialization
 - Seed - random number 고정
 - Actor & Critic network
 - Actor & Critic Optimizer
 - TensorboardX
 - Recent rewards

```
97  def main():
98      env = gym.make(args.env_name)
99      env.seed(500)
100     torch.manual_seed(500)
101
102     state_size = env.observation_space.shape[0]
103     action_size = env.action_space.shape[0]
104     print('state size:', state_size)
105     print('action size:', action_size)
106
107     actor = Actor(state_size, action_size, args)
108     critic = Critic(state_size, args)
109
110     actor_optimizer = optim.Adam(actor.parameters(), lr=args.actor_lr)
111     critic_optimizer = optim.Adam(critic.parameters(), lr=args.critic_lr)
112
113     writer = SummaryWriter(args.logdir)
114
115     recent_rewards = deque(maxlen=100)
116     episodes = 0
```



Main loop

- Episode 진행
 - Initialize trajectories set
 - 상태에 따른 행동 선택
 - 다음 상태와 보상을 받음
 - Trajectories set에 저장

```
118     for iter in range(args.max_iter_num):  
119         trajectories = deque()  
120         steps = 0  
121  
122         while steps < args.total_sample_size:  
123             done = False  
124             score = 0  
125             episodes += 1  
126  
127             state = env.reset()  
128             state = np.reshape(state, [1, state_size])  
129  
130             while not done:  
131                 if args.render:  
132                     env.render()  
133  
134                 steps += 1  
135  
136                 mu, std = actor(torch.Tensor(state))  
137                 action = get_action(mu, std)  
138  
139                 next_state, reward, done, _ = env.step(action)  
140  
141                 mask = 0 if done else 1  
142  
143                 trajectories.append((state, action, reward, mask))  
144  
145                 next_state = np.reshape(next_state, [1, state_size])  
146                 state = next_state  
147                 score += reward  
148  
149                 if done:  
150                     recent_rewards.append(score)
```



Main loop

- Train model
- Print & Visualize log
- Termination : 최근 100개의 episode의 평균 score가 -300보다 크다면
 - Save model
 - 학습 종료

```
152     actor.train(), critic.train()
153     train_model(actor, critic, actor_optimizer, critic_optimizer,
154                 | | |
155                 | | | trajectories, state_size, action_size)
156
157     writer.add_scalar('log/score', float(score), episodes)
158
159     if iter % args.log_interval == 0:
160         print('{} iter | {} episode | score_avg: {:.2f}'.format(iter, episodes, np.mean(recent_rewards)))
161
162     if np.mean(recent_rewards) > args.goal_score:
163         if not os.path.isdir(args.save_path):
164             os.makedirs(args.save_path)
165
166         ckpt_path = args.save_path + 'model.pth'
167         torch.save(actor.state_dict(), ckpt_path)
168         print('Recent rewards exceed -300. So end')
169         break
```

Train model

- Trajectories → Numpy array
- Trajectories에 있는 2200개의 sample들을 각각 나눔
 - state - (2200, 3)
 - action - (2200, 1)
 - reward - (2200)
 - mask - (2200)

```
35  def train_model(actor, critic, actor_optimizer, critic_optimizer,
36  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
37  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
38  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
39  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
40  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
41  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
42  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
43  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
44  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
45  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
```

Train model

- **returns** - (2200)
- **old_policy** - (2200, 1)

```
47     returns = get_returns(rewards, masks, args.gamma)
48
49     mu, std = actor(torch.Tensor(states))
50     old_policy = get_log_prob(actions, mu, std)
```

Train model

- `states_samples` - (64, 3)
- `actions_samples` - (64, 1)

```
52     criterion = torch.nn.MSELoss()
53
54     n = len(states)
55     arr = np.arange(n)
56
57     for _ in range(args.model_update_num):
58         np.random.shuffle(arr)
59
60         for i in range(n // args.batch_size):
61             mini_batch_index = arr[args.batch_size * i : args.batch_size * (i + 1)]
62             mini_batch_index = torch.LongTensor(mini_batch_index)
63
64             states_samples = torch.Tensor(states)[mini_batch_index]
65             actions_samples = torch.Tensor(actions)[mini_batch_index]
```

Train model

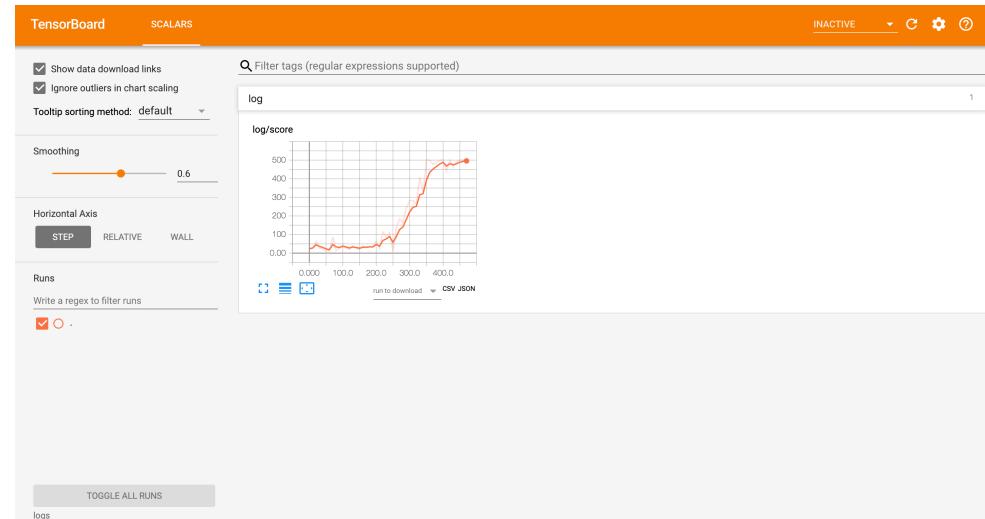
- **values_samples** - (64, 1)
- **targets_samples** - (64, 1)
- **ratio** - (64, 1)
- **advantages_samples** - (64, 1)
- **clipped_ratio** - (64, 1)

```
67 |         # get critic loss
68 |         values_samples = critic(states_samples)
69 |         targets_samples = returns.unsqueeze(1)[mini_batch_index]
70 |
71 |         critic_loss = criterion(values_samples, targets_samples)
72 |
73 |         # get actor loss
74 |         actor_loss, ratio, advantages_samples = surrogate_loss(actor, values_samples, targets_samples,
75 |                                         states_samples, old_policy.detach(),
76 |                                         actions_samples, mini_batch_index)
77 |
78 |         clipped_ratio = torch.clamp(ratio,
79 |                                         1.0 - args.clip_param,
80 |                                         1.0 + args.clip_param)
81 |         clipped_actor_loss = clipped_ratio * advantages_samples
82 |
83 |         actor_loss = -torch.min(actor_loss, clipped_actor_loss).mean()
84 |
85 |         # update actor & critic
86 |         loss = actor_loss + 0.5 * critic_loss
87 |
88 |         critic_optimizer.zero_grad()
89 |         loss.backward(retain_graph=True)
90 |         critic_optimizer.step()
91 |
92 |         actor_optimizer.zero_grad()
93 |         loss.backward()
94 |         actor_optimizer.step()
```



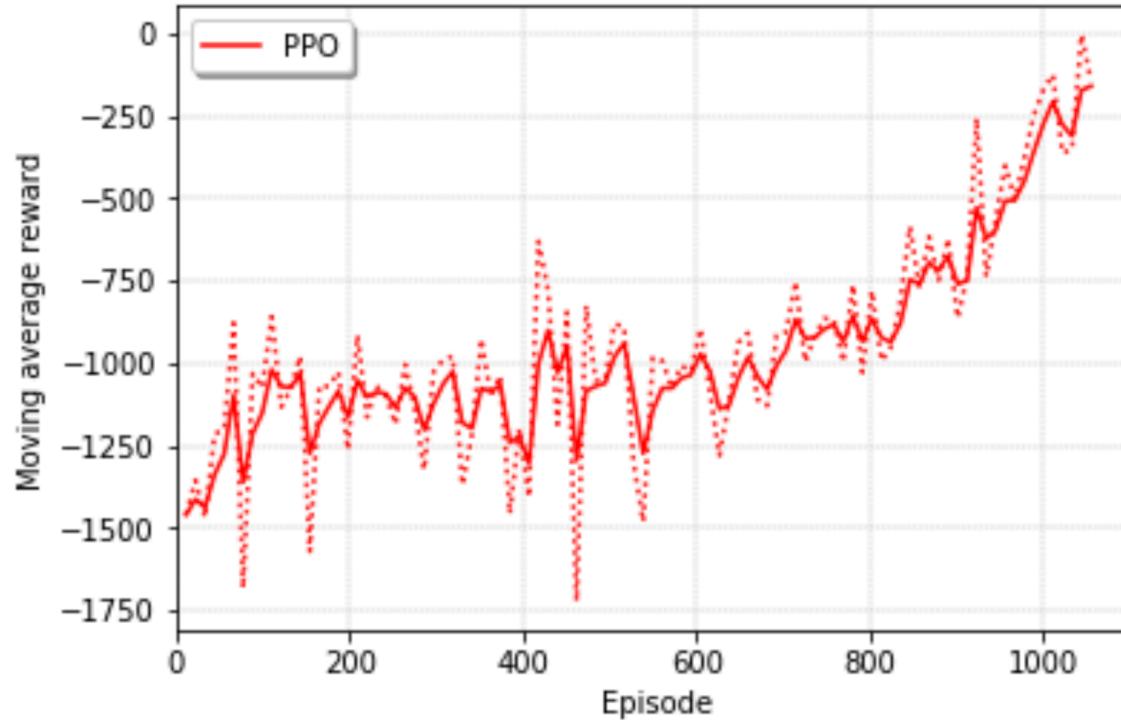
Train & TensorboardX

- 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

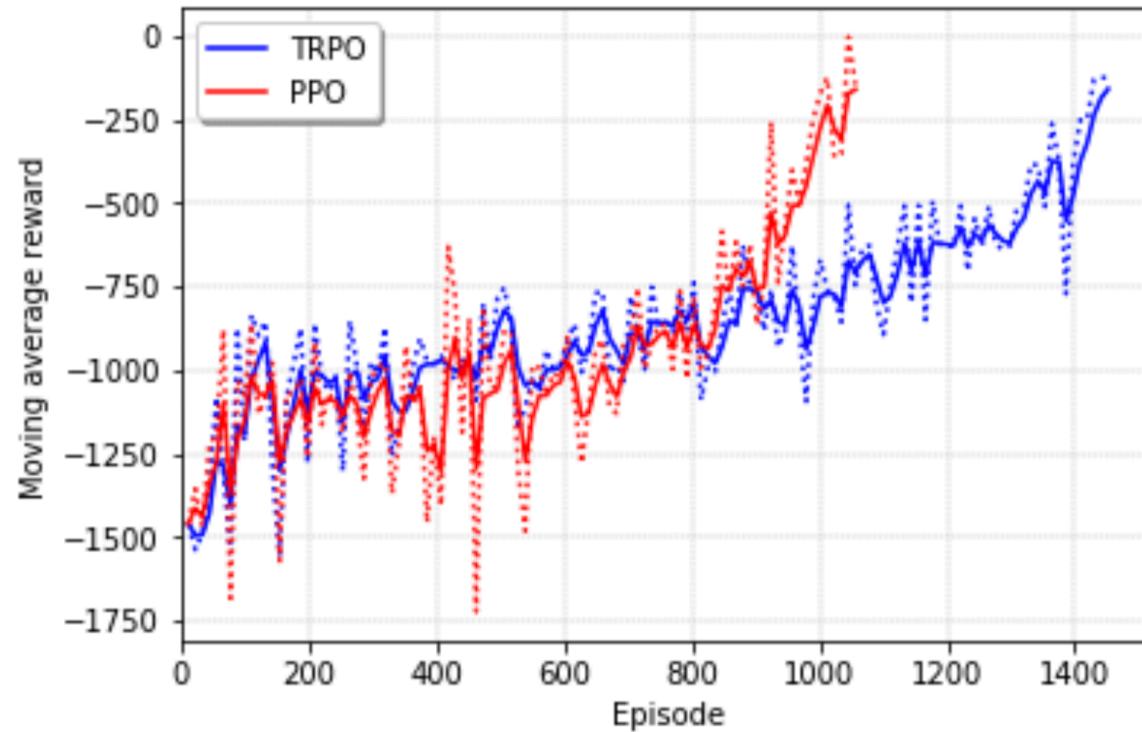
- Learning curve



- Test
 - `python test.py`

TRPO vs. PPO

- Learning curve



Thank you



CORE
Control + Optimization Research Lab