CORE
Control + Optimization Research Lab

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** $k = 0, 1, 2, \ldots$ **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:

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

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**

source : https://spinningup.openai.com/en/latest/algorithms/ppo.html

Proximal Policy Optimization (PPO)

- Learning process (PPO + GAE)
  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 : Return과 GAE 구하기
     - Step 2 : Mini batch 형태로 Actor & Critic network 업데이트

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
```

Learning process

1. 상태에 따른 행동 선택

```
146                 mu, std = actor(torch.Tensor(state))
147                 action = get_action(mu, std)
```

train.py

```
5   def get_action(mu, std):
6       normal = Normal(mu, std)
7       action = normal.sample()
8
9       return action.data.numpy()
```

utils.py

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

Learning process

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

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

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

```
129            trajectories = deque()
```

```
151                    mask = 0 if done else 1
152
153                    trajectories.append((state, action, reward, mask))
```

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
   - Step 1 : Return과 GAE 구하기

utils.py

```python
def get_gae(rewards, masks, values, args):
    returns = torch.zeros_like(rewards)
    advantages = torch.zeros_like(rewards)

    running_returns = 0
    previous_value = 0
    running_advants = 0

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

        # advantage
        running_deltas = rewards[t] + masks[t] * args.gamma * previous_value - values.data[t]
        running_advants = running_deltas + masks[t] * args.gamma * args.lamda * running_advants

        previous_value = values.data[t]
        advantages[t] = running_advants

    advantages = (advantages - advantages.mean()) / advantages.std()

    return returns, advantages
```

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
   - Step 2 : Mini batch 형태로 Actor & Critic network 업데이트
     - Mini batch로 나누기

train.py

```
53      criterion = torch.nn.MSELoss()
54
55      n = len(states)
56      arr = np.arange(n)
57
58      for _ in range(args.model_update_num):
59          np.random.shuffle(arr)
60
61          for i in range(n // args.batch_size):
62              mini_batch_index = arr[args.batch_size * i : args.batch_size * (i + 1)]
63              mini_batch_index = torch.LongTensor(mini_batch_index)
64
65              states_samples = torch.Tensor(states)[mini_batch_index]
66              actions_samples = torch.Tensor(actions)[mini_batch_index]
67              returns_samples = returns.unsqueeze(1)[mini_batch_index]
68              advantages_samples = advantages.unsqueeze(1)[mini_batch_index]
69              old_values_samples = old_values[mini_batch_index].detach()
```

```
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
```

- n = 2200
- Batch size = 64

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
   - Step 2 : Mini batch 형태로 Actor & Critic network 업데이트
     - Critic Loss (clip param : 0.2)

$$J_V(\phi) = \max\Big(\ (V_\phi(s) - R)^2, \ \ (V_{\phi_{old}}(s) + \text{clip}\big(V_\phi(s) - V_{\phi_{old}}(s), -\epsilon, +\epsilon\big) - R)^2 \ \Big)$$

train.py

```
71            # get critic loss
72            values_samples = critic(states_samples)
73            clipped_values_samples = old_values_samples + \
74                                     torch.clamp(values_samples - old_values_samples,
75                                                 -args.clip_param,
76                                                 args.clip_param)
77
78            critic_loss = criterion(values_samples, returns_samples)
79            clipped_critic_loss = criterion(clipped_values_samples, returns_samples)
80
81            critic_loss = torch.max(critic_loss, clipped_critic_loss)
```

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

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

- Main objective function of PPO

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

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)

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

train.py

```
83            # get actor loss
84            actor_loss, ratio = surrogate_loss(actor, advantages_samples, states_samples,
85                                               old_policy.detach(), actions_samples,
86                                               mini_batch_index)
87
88            clipped_ratio = torch.clamp(ratio,
89                                        1.0 - args.clip_param,
90                                        1.0 + args.clip_param)
91            clipped_actor_loss = clipped_ratio * advantages_samples
92
93            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)

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

utils.py

```
41  def surrogate_loss(actor, advantages, states, old_policy, actions, batch_index):
42      mu, std = actor(torch.Tensor(states))
43      new_policy = get_log_prob(actions, mu, std)
44
45      old_policy = old_policy[batch_index]
46
47      ratio = torch.exp(new_policy - old_policy)
48      surrogate_loss = ratio * advantages
49
50      return surrogate_loss, ratio
```

Learning process

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

train.py

```
95            # update actor & critic
96            loss = actor_loss + 0.5 * critic_loss
97
98            critic_optimizer.zero_grad()
99            loss.backward(retain_graph=True)
100           critic_optimizer.step()
101
102           actor_optimizer.zero_grad()
103           loss.backward()
104           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

```
107  def main():
108      env = gym.make(args.env_name)
109      env.seed(500)
110      torch.manual_seed(500)
111
112      state_size = env.observation_space.shape[0]
113      action_size = env.action_space.shape[0]
114      print('state size:', state_size)
115      print('action size:', action_size)
116
117      actor = Actor(state_size, action_size, args)
118      critic = Critic(state_size, args)
119
120      actor_optimizer = optim.Adam(actor.parameters(), lr=args.actor_lr)
121      critic_optimizer = optim.Adam(critic.parameters(), lr=args.critic_lr)
122
123      writer = SummaryWriter(args.logdir)
124
125      recent_rewards = deque(maxlen=100)
126      episodes = 0
```

Main loop

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

```
128     for iter in range(args.max_iter_num):
129         trajectories = deque()
130         steps = 0
131 
132         while steps < args.total_sample_size:
133             done = False
134             score = 0
135             episodes += 1
136 
137             state = env.reset()
138             state = np.reshape(state, [1, state_size])
139 
140             while not done:
141                 if args.render:
142                     env.render()
143 
144                 steps += 1
145 
146                 mu, std = actor(torch.Tensor(state))
147                 action = get_action(mu, std)
148 
149                 next_state, reward, done, _ = env.step(action)
150 
151                 mask = 0 if done else 1
152 
153                 trajectories.append((state, action, reward, mask))
154 
155                 next_state = np.reshape(next_state, [1, state_size])
156                 state = next_state
157                 score += reward
158 
159                 if done:
160                     recent_rewards.append(score)
```

Main loop

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

```
162         actor.train(), critic.train()
163         train_model(actor, critic, actor_optimizer, critic_optimizer,
164                     trajectories, state_size, action_size)
165
166         writer.add_scalar('log/score', float(score), episodes)
167
168         if iter % args.log_interval == 0:
169             print('{} iter | {} episode | score_avg: {:.2f}'.format(iter, episodes, np.mean(recent_rewards)))
170
171         if np.mean(recent_rewards) > args.goal_score:
172             if not os.path.isdir(args.save_path):
173                 os.makedirs(args.save_path)
174
175             ckpt_path = args.save_path + 'model.pth.tar'
176             torch.save(actor.state_dict(), ckpt_path)
177             print('Recent rewards exceed -300. So end')
178             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                  trajectories, state_size, action_size):
37      trajectories = np.array(trajectories)
38      states = np.vstack(trajectories[:, 0])
39      actions = list(trajectories[:, 1])
40      rewards = list(trajectories[:, 2])
41      masks = list(trajectories[:, 3])
42
43      actions = torch.Tensor(actions).squeeze(1)
44      rewards = torch.Tensor(rewards).squeeze(1)
45      masks = torch.Tensor(masks)
```

Train model

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

```
47      old_values = critic(torch.Tensor(states))
48      returns, advantages = get_gae(rewards, masks, old_values, args)
49
50      mu, std = actor(torch.Tensor(states))
51      old_policy = get_log_prob(actions, mu, std)
```

Train model

- states_samples - (64, 3)
- actions_samples - (64, 1)
- returns_samples - (64, 1)
- advantages_samples - (64, 1)
- old_values_samples - (64, 1)

```
53      criterion = torch.nn.MSELoss()
54
55      n = len(states)
56      arr = np.arange(n)
57
58      for _ in range(args.model_update_num):
59          np.random.shuffle(arr)
60
61          for i in range(n // args.batch_size):
62              mini_batch_index = arr[args.batch_size * i : args.batch_size * (i + 1)]
63              mini_batch_index = torch.LongTensor(mini_batch_index)
64
65              states_samples = torch.Tensor(states)[mini_batch_index]
66              actions_samples = torch.Tensor(actions)[mini_batch_index]
67              returns_samples = returns.unsqueeze(1)[mini_batch_index]
68              advantages_samples = advantages.unsqueeze(1)[mini_batch_index]
69              old_values_samples = old_values[mini_batch_index].detach()
```

Train model

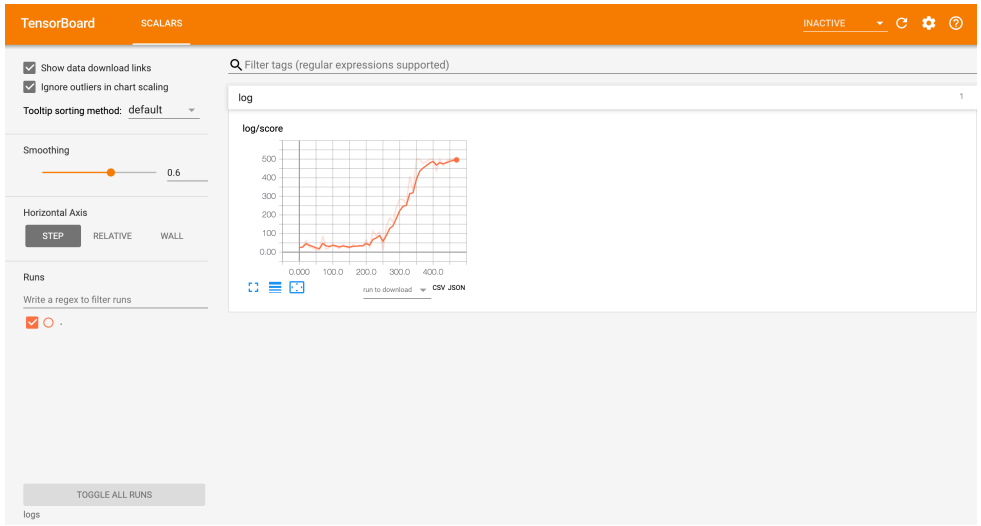
- **values_samples** - (64, 1)
- **clipped_values_samples** - (64, 1)
- **ratio** - (64, 1)
- **clipped_ratio** - (64, 1)

```
71          # get critic loss
72          values_samples = critic(states_samples)
73          clipped_values_samples = old_values_samples + \
74                                   torch.clamp(values_samples - old_values_samples,
75                                               -args.clip_param,
76                                               args.clip_param)
77
78          critic_loss = criterion(values_samples, returns_samples)
79          clipped_critic_loss = criterion(clipped_values_samples, returns_samples)
80
81          critic_loss = torch.max(critic_loss, clipped_critic_loss)
82
83          # get actor loss
84          actor_loss, ratio = surrogate_loss(actor, advantages_samples, states_samples,
85                                             old_policy.detach(), actions_samples,
86                                             mini_batch_index)
87
88          clipped_ratio = torch.clamp(ratio,
89                                      1.0 - args.clip_param,
90                                      1.0 + args.clip_param)
91          clipped_actor_loss = clipped_ratio * advantages_samples
92
93          actor_loss = -torch.min(actor_loss, clipped_actor_loss).mean()
94
95          # update actor & critic
96          loss = actor_loss + 0.5 * critic_loss
97
98          critic_optimizer.zero_grad()
99          loss.backward(retain_graph=True)
100         critic_optimizer.step()
101
102         actor_optimizer.zero_grad()
103         loss.backward()
104         actor_optimizer.step()
```

Train & TensorboardX

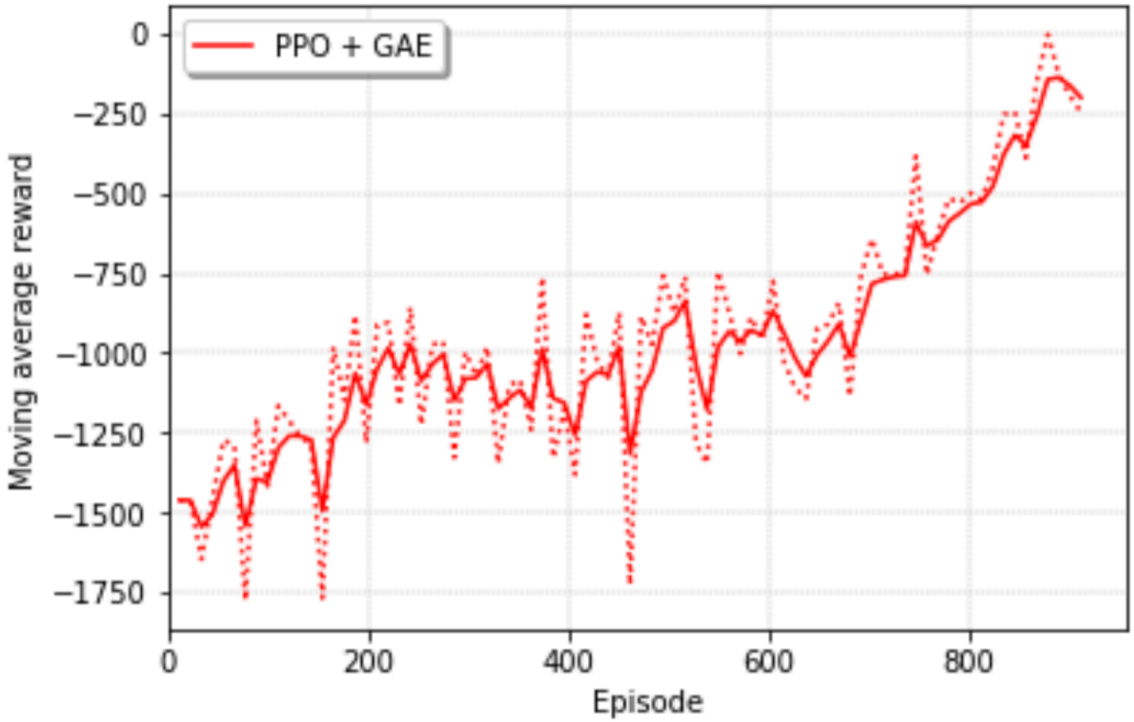
- Terminal A - train 실행
  - conda activate env_name
  - python train.py

- Terminal B - tensorboardX 실행
  - conda activate env_name
  - tensorboard --logdir logs
  - (웹에서) localhost:6006



Learning curve & Test

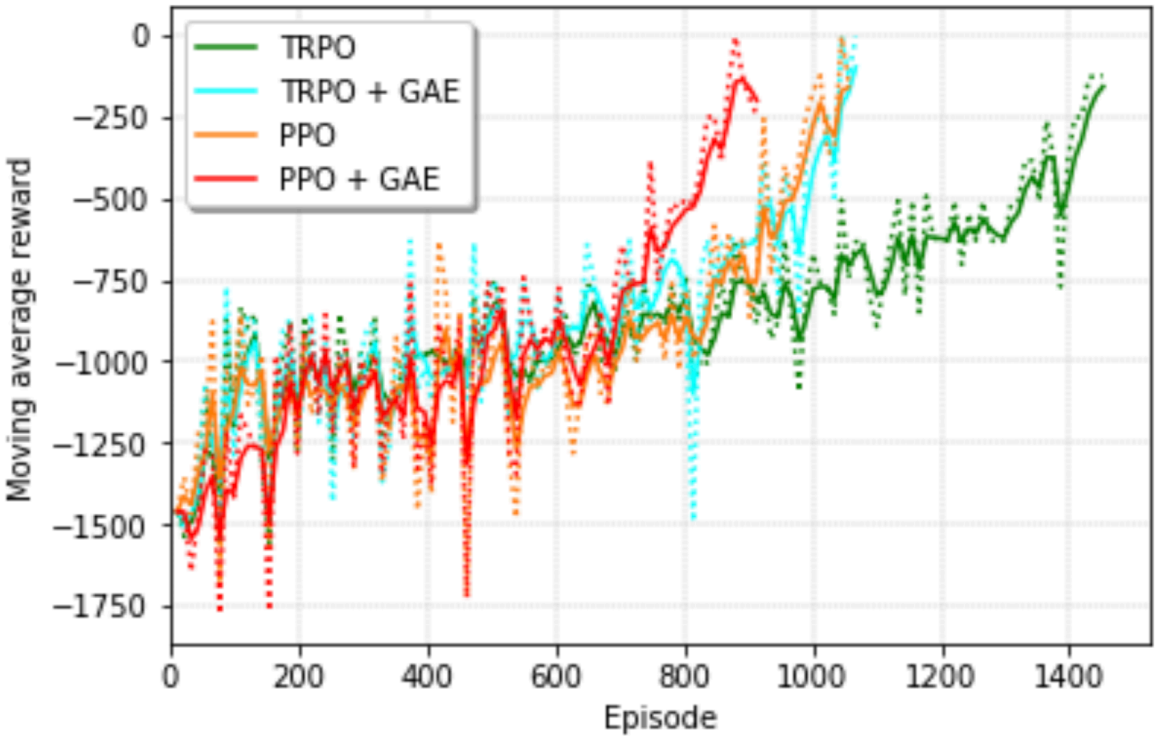
- Learning curve



- Test
  - ➢ python test.py

TRPO vs. PPO

- Learning curve



Thank you