CORE
Control + Optimization Research Lab

Proximal Policy Optimization (PPO)

이동민

삼성전자 서울대 공동연구소
Jul 19, 2019

Outline

- Proximal Policy Optimization (PPO)
  - Learning process
  - Hyperparameter
  - Main loop
  - Train model
  - Train & TensorboardX
  - Learning curve & Test
- TRPO vs. TRPO+GAE vs. PPO+GAE - Learning curve
- Comparison of algorithms for continuous action - 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**

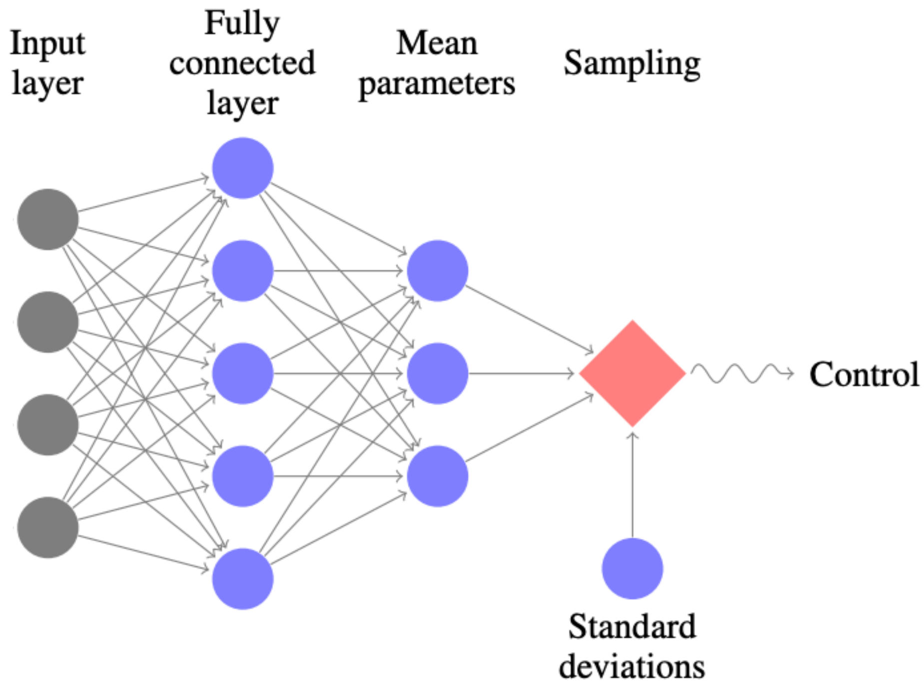
source : https://spinningup.openai.com/en/latest/algorithms/ppo.html

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, s')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

Trust Region Policy Optimization (TRPO)

- Actor network



Trust Region Policy Optimization (TRPO)

- Actor network

```python
class Actor(nn.Module):
    def __init__(self, state_size, action_size, args):
        super(Actor, self).__init__()
        self.fc1 = nn.Linear(state_size, args.hidden_size)
        self.fc2 = nn.Linear(args.hidden_size, args.hidden_size)
        self.fc3 = nn.Linear(args.hidden_size, action_size)

    def forward(self, x):
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))

        mu = self.fc3(x)
        log_std = torch.zeros_like(mu)
        std = torch.exp(log_std)

        return mu, std
```

Trust Region Policy Optimization (TRPO)

- Critic network

```python
class Critic(nn.Module):
    def __init__(self, state_size, args):
        super(Critic, self).__init__()
        self.fc1 = nn.Linear(state_size, args.hidden_size)
        self.fc2 = nn.Linear(args.hidden_size, args.hidden_size)
        self.fc3 = nn.Linear(args.hidden_size, 1)

    def forward(self, x):
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        value = self.fc3(x)

        return value
```

Learning process

1. 상태에 따른 행동 선택

```
mu, std = actor(torch.Tensor(state))
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

Learning process

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

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

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

```
trajectories = deque()
```

```
mask = 0 if done else 1

trajectories.append((state, action, reward, mask))
```

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

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

Learning process

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

```
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

```
criterion = torch.nn.MSELoss()

n = len(states)
arr = np.arange(n)

for _ in range(args.model_update_num):
    np.random.shuffle(arr)

    for i in range(n // args.batch_size):
        mini_batch_index = arr[args.batch_size * i : args.batch_size * (i + 1)]
        mini_batch_index = torch.LongTensor(mini_batch_index)

        states_samples = torch.Tensor(states)[mini_batch_index]
        actions_samples = torch.Tensor(actions)[mini_batch_index]
        returns_samples = returns.unsqueeze(1)[mini_batch_index]
        advantages_samples = advantages.unsqueeze(1)[mini_batch_index]
        old_values_samples = old_values[mini_batch_index].detach()
```

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)

$$J_V(\theta) = \max\Big(\ (V_\theta(s) - R)^2,\ \ (V_{\theta_{old}}(s) + \mathrm{clip}\big(V_\theta(s) - V_{\theta_{old}}(s), -\epsilon, +\epsilon\big) - R)^2 \ \Big)$$

```
# get critic loss
values_samples = critic(states_samples)
clipped_values_samples = old_values_samples + \
                         torch.clamp(values_samples - old_values_samples,
                                     -args.clip_param,
                                     args.clip_param)

critic_loss = criterion(values_samples, returns_samples)
clipped_critic_loss = criterion(clipped_values_samples, returns_samples)

critic_loss = torch.max(critic_loss, clipped_critic_loss)
```

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)

$$J_\pi(\theta) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t,\ \mathrm{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) \hat{A}_t \right)$$

```
# get actor loss
actor_loss, ratio = surrogate_loss(actor, advantages_samples, states_samples,
                                   old_policy.detach(), actions_samples,
                                   mini_batch_index)

clipped_ratio = torch.clamp(ratio,
                            1.0 - args.clip_param,
                            1.0 + args.clip_param)
clipped_actor_loss = clipped_ratio * advantages_samples

actor_loss = -torch.min(actor_loss, clipped_actor_loss).mean()
```

```
def surrogate_loss(actor, advantages, 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)
    surrogate_loss = ratio * advantages

    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 & ciritic

```
# update actor & critic
loss = actor_loss + 0.5 * critic_loss

critic_optimizer.zero_grad()
loss.backward(retain_graph=True)
critic_optimizer.step()

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

Hyperparameter

```python
parser = argparse.ArgumentParser()
parser.add_argument('--env_name', type=str, default="Pendulum-v0")
parser.add_argument('--load_model', type=str, default=None)
parser.add_argument('--save_path', default='./save_model/', help='')
parser.add_argument('--render', action="store_true", default=False)
parser.add_argument('--gamma', type=float, default=0.99)
parser.add_argument('--lamda', type=float, default=0.98)
parser.add_argument('--hidden_size', type=int, default=64)
parser.add_argument('--batch_size', type=int, default=64)
parser.add_argument('--actor_lr', type=float, default=1e-3)
parser.add_argument('--critic_lr', type=float, default=1e-3)
parser.add_argument('--model_update_num', type=int, default=10)
parser.add_argument('--clip_param', type=float, default=0.2)
parser.add_argument('--max_iter_num', type=int, default=500)
parser.add_argument('--total_sample_size', type=int, default=2048)
parser.add_argument('--log_interval', type=int, default=5)
parser.add_argument('--goal_score', type=int, default=-300)
parser.add_argument('--logdir', type=str, default='./logs',
                    help='tensorboardx logs directory')
args = parser.parse_args()
```

Main loop

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

```python
def main():
    env = gym.make(args.env_name)
    env.seed(500)
    torch.manual_seed(500)

    state_size = env.observation_space.shape[0]
    action_size = env.action_space.shape[0]
    print('state size:', state_size)
    print('action size:', action_size)

    actor = Actor(state_size, action_size, args)
    critic = Critic(state_size, args)

    actor_optimizer = optim.Adam(actor.parameters(), lr=args.actor_lr)
    critic_optimizer = optim.Adam(critic.parameters(), lr=args.critic_lr)

    writer = SummaryWriter(args.logdir)

    recent_rewards = deque(maxlen=100)
    episodes = 0
```

Main loop

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

```
for iter in range(args.max_iter_num):
    trajectories = deque()
    steps = 0

    while steps < args.total_sample_size:
        done = False
        score = 0
        episodes += 1

        state = env.reset()
        state = np.reshape(state, [1, state_size])

        while not done:
            if args.render:
                env.render()

            steps += 1

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

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

            mask = 0 if done else 1

            trajectories.append((state, action, reward, mask))

            next_state = np.reshape(next_state, [1, state_size])
            state = next_state
            score += reward

            if done:
                recent_rewards.append(score)
```

Main loop

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

```
actor.train(), critic.train()
train_model(actor, critic, actor_optimizer, critic_optimizer,
            trajectories, state_size, action_size)

writer.add_scalar('log/score', float(score), episodes)

if iter % args.log_interval == 0:
    print('{} iter | {} episode | score_avg: {:.2f}'.format(iter, episodes, np.mean(recent_rewards)))

if np.mean(recent_rewards) > args.goal_score:
    if not os.path.isdir(args.save_path):
        os.makedirs(args.save_path)

    ckpt_path = args.save_path + 'model.pth'
    torch.save(actor.state_dict(), ckpt_path)
    print('Recent rewards exceed -300. So end')
    break
```

Train model

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

```python
def train_model(actor, critic, actor_optimizer, critic_optimizer,
                trajectories, state_size, action_size):
    trajectories = np.array(trajectories)
    states = np.vstack(trajectories[:, 0])
    actions = list(trajectories[:, 1])
    rewards = list(trajectories[:, 2])
    masks = list(trajectories[:, 3])

    actions = torch.Tensor(actions).squeeze(1)
    rewards = torch.Tensor(rewards).squeeze(1)
    masks = torch.Tensor(masks)
```

Train model

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

```
old_values = critic(torch.Tensor(states))
returns, advantages = get_gae(rewards, masks, old_values, args)

mu, std = actor(torch.Tensor(states))
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_value_samples -** (64, 1)

```
criterion = torch.nn.MSELoss()

n = len(states)
arr = np.arange(n)

for _ in range(args.model_update_num):
    np.random.shuffle(arr)

    for i in range(n // args.batch_size):
        mini_batch_index = arr[args.batch_size * i : args.batch_size * (i + 1)]
        mini_batch_index = torch.LongTensor(mini_batch_index)

        states_samples = torch.Tensor(states)[mini_batch_index]
        actions_samples = torch.Tensor(actions)[mini_batch_index]
        returns_samples = returns.unsqueeze(1)[mini_batch_index]
        advantages_samples = advantages.unsqueeze(1)[mini_batch_index]
        old_values_samples = old_values[mini_batch_index].detach()
```

Train model

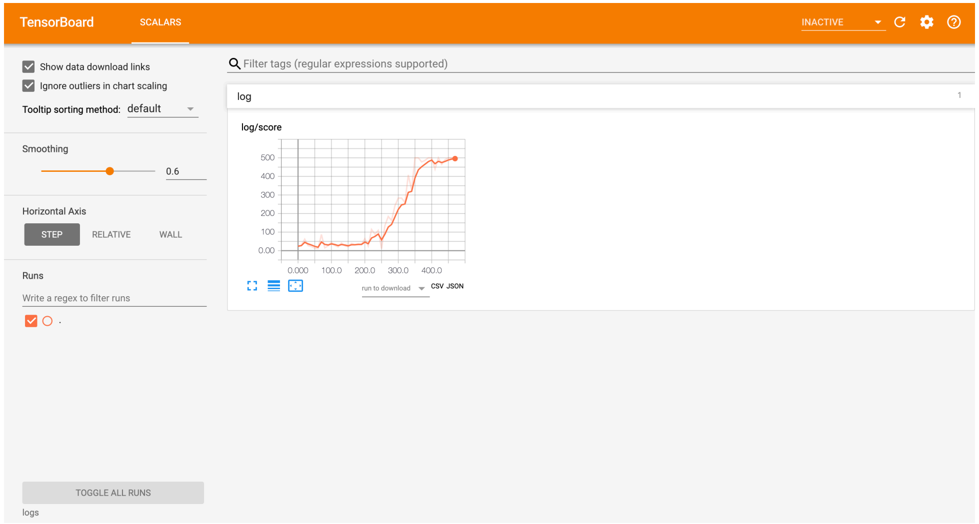
- **values_samples -** (64, 1)
- **clipped_values_samples -** (64, 1)
- **ratio -** (64, 1)
- **clipped_ratio -** (64, 1)

```python
# get critic loss
values_samples = critic(states_samples)
clipped_values_samples = old_values_samples + \
                         torch.clamp(values_samples - old_values_samples,
                                     -args.clip_param,
                                     args.clip_param)

critic_loss = criterion(values_samples, returns_samples)
clipped_critic_loss = criterion(clipped_values_samples, returns_samples)

critic_loss = torch.max(critic_loss, clipped_critic_loss)

# get actor loss
actor_loss, ratio = surrogate_loss(actor, advantages_samples, states_samples,
                                   old_policy.detach(), actions_samples,
                                   mini_batch_index)

clipped_ratio = torch.clamp(ratio,
                            1.0 - args.clip_param,
                            1.0 + args.clip_param)
clipped_actor_loss = clipped_ratio * advantages_samples

actor_loss = -torch.min(actor_loss, clipped_actor_loss).mean()

# update actor & critic
loss = actor_loss + 0.5 * critic_loss

critic_optimizer.zero_grad()
loss.backward(retain_graph=True)
critic_optimizer.step()

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

Train & TensorboardX

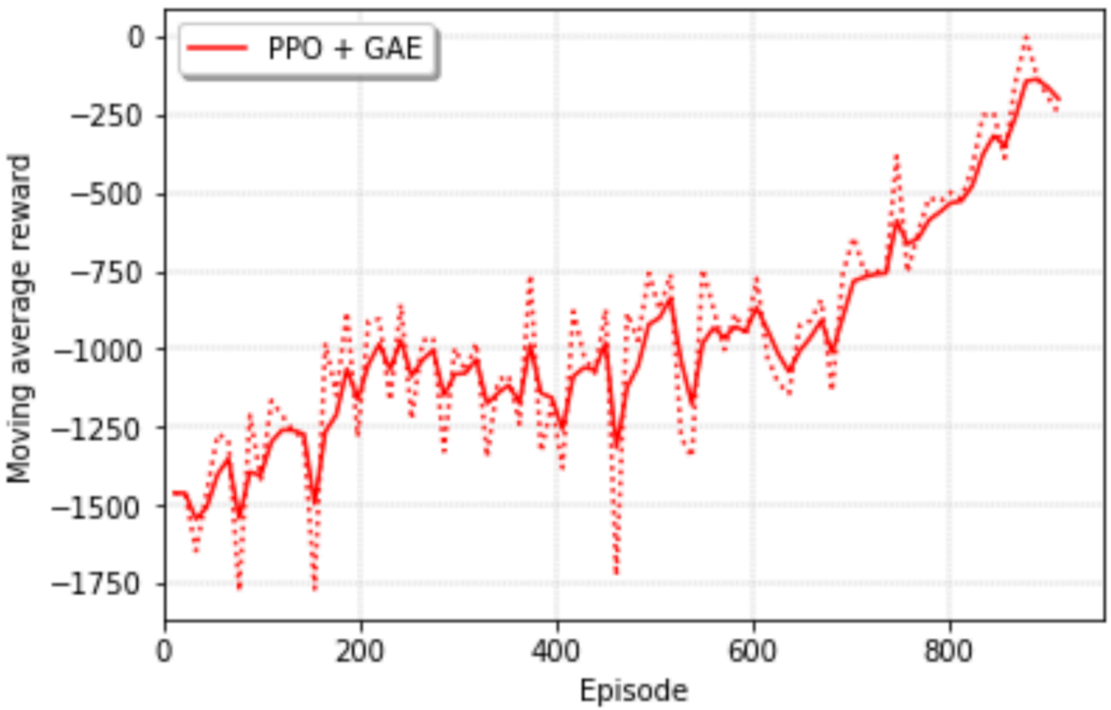
- Terminal A - train 실행
  - ➢ conda activate env_name
  - ➢ python train.py

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



Learning curve & Test

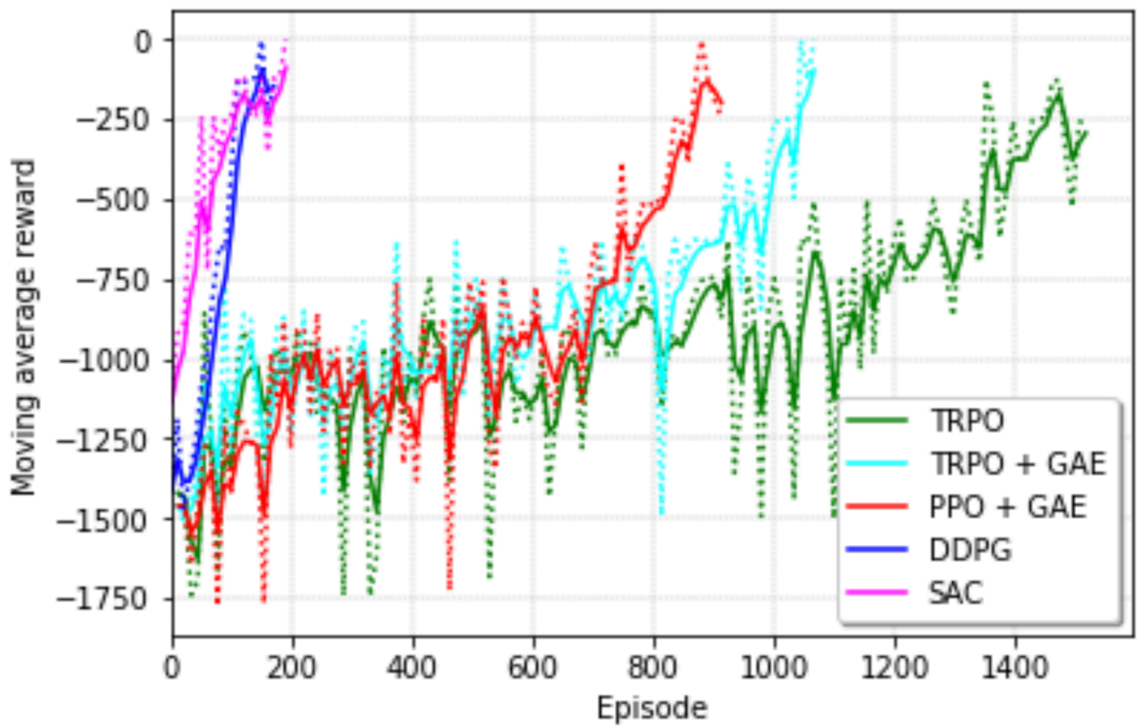
- Learning curve



- Test
  - python test.py

Comparison of algorithms for continuous action

- Learning curve



Comparison of algorithms for continuous action

Continuous action일 때는 어떤 알고리즘이 제일 좋을까요? → SAC
그리고 그 알고리즘이 왜 좋을까요?

- TRPO, PPO (On-policy algorithms)
  - **Poor sample efficiency** → 업데이트를 하기 위해서는 새로운 sample들을 수집해야 함
  - The number of gradient step and samples per step that are **extravagantly expensive**
- DDPG (Off-policy algorithm)
  - The interplay between the **deterministic actor** and the Q-function typically makes DDPG extreme **difficult to stabilize and brittle to hyperparameter settings**
  - Exploration noise can cause **sudden failures in unstable environments**
- SAC (Off-policy algorithm)
  - The **stochastic actor** aims to maximize expected reward while also **maximizing entropy**
  - **Automatic** gradient-based **temperature hyperparameter tuning** method

Thank you