

Generalized Advantage Estimator (GAE)

이동민

삼성전자 서울대 공동연구소
Jul 19, 2019

Outline

- Generalized Advantage Estimator (GAE)
 - Learning process
 - Hyperparameter
 - Main loop
 - Train model
 - Train & TensorboardX
 - Learning curve & Test
- TRPO vs. TRPO + GAE - Learning curve

Generalized Advantage Estimator (GAE)

- Learning process (TRPO + 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 : Critic network 업데이트
 - Step 3 : Surrogate loss를 통해 actor loss의 gradient 구하기
 - Step 4 : Conjugate gradient method를 통해 search direction 구하기
 - Step 5 : Step size와 maximal step 구하기
 - Step 6 : Backtracking line search를 통해 trust region안에서 actor network 업데이트

Generalized Advantage Estimator (GAE)

- 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
```

Generalized Advantage Estimator (GAE)

- 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
```

Learning process

1. 상태에 따른 행동 선택

```
141     mu, std = actor(torch.Tensor(state)) train.py  
142     action = get_action(mu, std)
```

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

- `Normal(mu, std)`
 - `Normal(Gaussian) distribution`에서 sampling을 할 경우, 분산을 일정하게 유지하면서 지속적인 exploration이 가능
 - `std`를 1로 고정함으로써 일정한 폭을 가지는 normal distribution에서 sampling

Learning process

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

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

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

```
124 | | | | trajectories = deque()  
146 | | | | mask = 0 if done else 1  
147 | | | |  
148 | | | | trajectories.append((state, action, reward, mask))
```

Learning process

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

- Step 1 : Return과 GAE 구하기

- time step 6까지 진행하고 episode가 끝났을 경우를 가정

$$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \gamma^3 R_5 + \gamma^4 R_6$$

$$G_2 = R_3 + \gamma R_4 + \gamma^2 R_5 + \gamma^3 R_6$$

$$G_3 = R_4 + \gamma R_5 + \gamma^2 R_6$$

$$G_4 = R_5 + \gamma R_6$$

$$G_5 = R_6$$

- 거꾸로 계산하며 계산해놓은 return값을 이용

$$G_5 = R_6$$

$$G_4 = R_5 + \gamma G_5$$

$$G_3 = R_4 + \gamma G_4$$

$$G_2 = R_3 + \gamma G_3$$

$$G_1 = R_2 + \gamma G_2$$

Learning process

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

- Step 1 : Return과 GAE 구하기
 - Generalized advantage estimator (GAE)

$$\hat{A}_t^{\text{GAE}(\gamma, \lambda)} = \sum_{\ell=0}^{\infty} (\gamma \lambda)^\ell \delta_{t+\ell}^V$$

$$\delta_t^V = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) = \hat{A}_t^\pi$$

- time step 5까지 진행하고 episode가 끝났을 경우를 가정

$$\hat{A}_1^{(1)} = \delta_1^V$$

$$\hat{A}_1^{(2)} = \delta_1^V + \gamma \lambda \delta_2^V$$

$$\hat{A}_1^{(3)} = \delta_1^V + \gamma \lambda \delta_2^V + \gamma^2 \lambda^2 \delta_3^V$$

$$\hat{A}_1^{(4)} = \delta_1^V + \gamma \lambda \delta_2^V + \gamma^2 \lambda^2 \delta_3^V + \gamma^3 \lambda^3 \delta_4^V$$

$$\hat{A}_1^{(5)} = \delta_1^V + \gamma \lambda \delta_2^V + \gamma^2 \lambda^2 \delta_3^V + \gamma^3 \lambda^3 \delta_4^V + \gamma^4 \lambda^4 \delta_5^V$$

Learning process

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

- Step 1 : Return과 GAE 구하기
 - 최종적으로 구하고 싶은 GAE

$$\hat{A}_1^{(5)} = \delta_1^V + \gamma\lambda\delta_2^V + \gamma^2\lambda^2\delta_3^V + \gamma^3\lambda^3\delta_4^V + \gamma^4\lambda^4\delta_5^V$$

- 거꾸로 계산하며 계산해놓은 delta값을 이용

$$\delta_5^V = r_5 - V^\pi(s_5) \quad \delta_5^V$$

$$\delta_4^V = r_4 + \gamma V^\pi(s_5) - V^\pi(s_4) \quad \delta_4^V + \gamma\lambda\delta_5^V$$

$$\delta_3^V = r_3 + \gamma V^\pi(s_4) - V^\pi(s_3) \quad \delta_3^V + \gamma\lambda\delta_4^V + \gamma^2\lambda^2\delta_5^V$$

$$\delta_2^V = r_2 + \gamma V^\pi(s_3) - V^\pi(s_2) \quad \delta_2^V + \gamma\lambda\delta_3^V + \gamma^2\lambda^2\delta_4^V + \gamma^3\lambda^3\delta_5^V$$

$$\delta_1^V = r_1 + \gamma V^\pi(s_2) - V^\pi(s_1) \quad \delta_1^V + \gamma\lambda\delta_2^V + \gamma^2\lambda^2\delta_3^V + \gamma^3\lambda^3\delta_4^V + \gamma^4\lambda^4\delta_5^V$$

Learning process

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

- Step 1 : Return과 GAE 구하기

```
11  def get_gae(rewards, masks, values, args):
12      returns = torch.zeros_like(rewards)
13      advantages = torch.zeros_like(rewards)
14
15      running_returns = 0
16      previous_value = 0
17      running_advants = 0
18
19      for t in reversed(range(0, len(rewards))):
20          # return
21          running_returns = rewards[t] + masks[t] * args.gamma * running_returns
22          returns[t] = running_returns
23
24          # advantage
25          running_deltas = rewards[t] + masks[t] * args.gamma * previous_value - values.data[t]
26          running_advants = running_deltas + masks[t] * args.gamma * args.lamda * running_advants
27
28          previous_value = values.data[t]
29          advantages[t] = running_advants
30
31      advantages = (advantages - advantages.mean()) / advantages.std()
32
33  return returns, advantages
```

utils.py

Learning process

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

- Step 2 : Critic network 업데이트

- Critic Loss

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

- $n = 2200$

- $\text{Batch size} = 64$

```
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
```

```
49      # -----
50      # step 2: update critic
51      criterion = torch.nn.MSELoss()
52
53      n = len(states)
54      arr = np.arange(n)
55
56      for _ in range(5):
57          np.random.shuffle(arr)
58
59          for i in range(n // args.batch_size):
60              mini_batch_index = arr[args.batch_size * i : args.batch_size * (i + 1)]
61              mini_batch_index = torch.LongTensor(mini_batch_index)
62
63              states_samples = torch.Tensor(states)[mini_batch_index]
64              values_samples = critic(states_samples)
65
66              target_samples = returns.unsqueeze(1)[mini_batch_index]
67
68              critic_loss = criterion(values_samples, target_samples)
69              critic_optimizer.zero_grad()
70              critic_loss.backward()
71              critic_optimizer.step()
```

train.py

Learning process

❖ 정리

1) Get surrogate objective function

$$\mathbb{E}_{s,a \sim \pi_{\theta_{old}}} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \right]$$

2) Find search direction, step size and maximal step through CGM and Hessian of KL

- Search direction : $H^{-1}g$
- Step size : $\sqrt{\frac{2\delta}{g^T H^{-1} g}}$
- Maximal step : $\sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1}g$

3) Do line search on that direction inside trust region through Backtracking line search

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1}g$$

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
 - Step 3 : Surrogate loss를 통해 actor loss의 gradient 구하기

```
73     # -----
74     # step 3: get gradient of actor loss through surrogate loss
75     mu, std = actor(torch.Tensor(states))
76     old_policy = get_log_prob(actions, mu, std)
```

train.py

```
35     def get_log_prob(actions, mu, std):      utils.py
36         normal = Normal(mu, std)
37         log_prob = normal.log_prob(actions)
38
39     return log_prob
```

$$\pi_{\theta}(a|s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - \mu_{\theta}(s))^2}{2\sigma^2}\right)$$

- Multiply the log on both sides

$$\log \pi_{\theta}(a|s) = -\log \sqrt{2\pi} - \log \sigma - \frac{(a - \mu_{\theta}(s))^2}{2\sigma^2}$$

Learning process

- 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
 - Step 3 : Surrogate loss를 통해 actor loss의 gradient 구하기

```
73     # -----
74     # step 3: get gradient of actor loss through surrogate loss      train.py
75     mu, std = actor(torch.Tensor(states))
76     old_policy = get_log_prob(actions, mu, std)
77     actor_loss = surrogate_loss(actor, advantages, states, old_policy.detach(), actions)

41     def surrogate_loss(actor, advantages, states, old_policy, actions):    utils.py
42         mu, std = actor(torch.Tensor(states))
43         new_policy = get_log_prob(actions, mu, std)
44
45         advantages = advantages.unsqueeze(1)
46
47         surrogate_loss = torch.exp(new_policy - old_policy) * advantages
48         surrogate_loss = surrogate_loss.mean()
49
50     return surrogate_loss
```

- surrogate_loss

$$\mathcal{L}_{\theta_{old}}(\theta) = \mathbb{E}_{s,a \sim \pi_{\theta_{old}}} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \right]$$

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
 - Step 3 : Surrogate loss를 통해 actor loss의 gradient 구하기

```
73     # -----
74     # step 3: get gradient of actor loss through surrogate loss
75     mu, std = actor(torch.Tensor(states))
76     old_policy = get_log_prob(actions, mu, std)
77     actor_loss = surrogate_loss(actor, advantages, states, old_policy.detach(), actions)
78
79     actor_loss_grad = torch.autograd.grad(actor_loss, actor.parameters())
80     actor_loss_grad = flat_grad(actor_loss_grad)
```

train.py

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
 - Step 4 : Conjugate gradient method를 통해 search direction 구하기

```
82      # -----
83      # step 4: get search direction through conjugate gradient method
84      search_dir = conjugate_gradient(actor, states, actor_loss_grad.data, nsteps=10)    train.py
```

Learning process

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

- Step 5 : Step size와 maximal step 구하기

train.py

```
86      # -----
87      # step 5: get step size and maximal step
88      gHg = (hessian_vector_product(actor, states, search_dir) * search_dir).sum(0, keepdim=True)
89      step_size = torch.sqrt(2 * args.max_kl / gHg)[0]
90      maximal_step = step_size * search_dir
```

- Step size : $\sqrt{\frac{2\delta}{g^T H^{-1} g}}$ ($Hx = g$), (max_kl δ : 0.01)
- Maximal step : $\sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$

Learning process

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

- Step 6 : Backtracking line search를 통해 trust region안에서 actor network 업데이트

```
92      # -----
93      # step 6: perform backtracking line search and update actor in trust region
94      params = flat_params(actor)
95
96      old_actor = Actor(state_size, action_size, args)
97      update_model(old_actor, params)
98
99      backtracking_line_search(old_actor, actor, actor_loss, actor_loss_grad,
100                                old_policy, params, maximal_step, args.max_kl,
101                                advantages, states, actions)
```

train.py

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('--critic_lr', type=float, default=1e-3)
24 parser.add_argument('--max_kl', type=float, default=1e-2)
25 parser.add_argument('--max_iter_num', type=int, default=500)
26 parser.add_argument('--total_sample_size', type=int, default=2048)
27 parser.add_argument('--log_interval', type=int, default=5)
28 parser.add_argument('--goal_score', type=int, default=-300)
29 parser.add_argument('--logdir', type=str, default='./logs',
30 |   |   |   |   |   help='tensorboardx logs directory')
31 args = parser.parse_args()
```



Main loop

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

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



Main loop

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

```
123     for iter in range(args.max_iter_num):
124         trajectories = deque()
125         steps = 0
126
127         while steps < args.total_sample_size:
128             done = False
129             score = 0
130             episodes += 1
131
132             state = env.reset()
133             state = np.reshape(state, [1, state_size])
134
135             while not done:
136                 if args.render:
137                     env.render()
138
139                 steps += 1
140
141                 mu, std = actor(torch.Tensor(state))
142                 action = get_action(mu, std)
143
144                 next_state, reward, done, _ = env.step(action)
145
146                 mask = 0 if done else 1
147
148                 trajectories.append((state, action, reward, mask))
149
150                 next_state = np.reshape(next_state, [1, state_size])
151                 state = next_state
152                 score += reward
153
154                 if done:
155                     recent_rewards.append(score)
```

Main loop

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

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

Train model

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

```
33 def train_model(actor, critic, critic_optimizer, trajectories, state_size, action_size):  
34     trajectories = np.array(trajectories)  
35     states = np.vstack(trajectories[:, 0])  
36     actions = list(trajectories[:, 1])  
37     rewards = list(trajectories[:, 2])  
38     masks = list(trajectories[:, 3])  
39  
40     actions = torch.Tensor(actions).squeeze(1)  
41     rewards = torch.Tensor(rewards).squeeze(1)  
42     masks = torch.Tensor(masks)
```

Train model

- **values** - (2200, 1)
- **returns** - (2200)
- **advantages** - (2200)
- **n** - 2200
- **batch_size** - 64

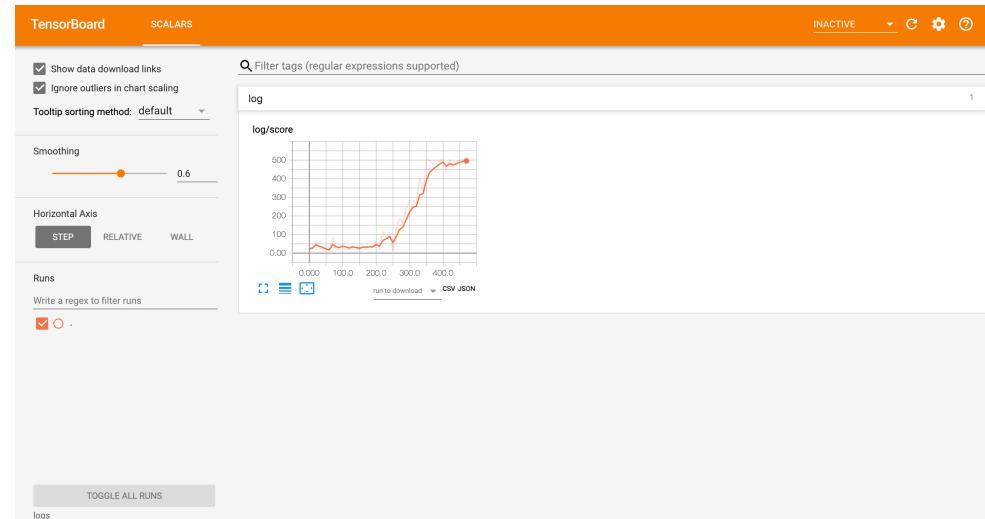
```
44     # -----
45     # step 1: get returns and GAEs
46     values = critic(torch.Tensor(states))
47     returns, advantages = get_gae(rewards, masks, values, args)
48
49     # -----
50     # step 2: update critic
51     criterion = torch.nn.MSELoss()
52
53     n = len(states)
54     arr = np.arange(n)
55
56     for _ in range(5):
57         np.random.shuffle(arr)
58
59         for i in range(n // args.batch_size):
60             mini_batch_index = arr[args.batch_size * i : args.batch_size * (i + 1)]
61             mini_batch_index = torch.LongTensor(mini_batch_index)
62
63             states_samples = torch.Tensor(states)[mini_batch_index]
64             values_samples = critic(states_samples)
65
66             target_samples = returns.unsqueeze(1)[mini_batch_index]
67
68             critic_loss = criterion(values_samples, target_samples)
69             critic_optimizer.zero_grad()
70             critic_loss.backward()
71             critic_optimizer.step()
```

Train model

```
73      # -----
74      # step 3: get gradient of actor loss through surrogate loss
75      mu, std = actor(torch.Tensor(states))
76      old_policy = get_log_prob(actions, mu, std)
77      actor_loss = surrogate_loss(actor, advantages, states, old_policy.detach(), actions)
78
79      actor_loss_grad = torch.autograd.grad(actor_loss, actor.parameters())
80      actor_loss_grad = flat_grad(actor_loss_grad)
81
82      # -----
83      # step 4: get search direction through conjugate gradient method
84      search_dir = conjugate_gradient(actor, states, actor_loss_grad.data, nsteps=10)
85
86      # -----
87      # step 5: get step size and maximal step
88      gHg = (hessian_vector_product(actor, states, search_dir) * search_dir).sum(0, keepdim=True)
89      step_size = torch.sqrt(2 * args.max_kl / gHg)[0]
90      maximal_step = step_size * search_dir
91
92      # -----
93      # step 6: perform backtracking line search and update actor in trust region
94      params = flat_params(actor)
95
96      old_actor = Actor(state_size, action_size, args)
97      update_model(old_actor, params)
98
99      backtracking_line_search(old_actor, actor, actor_loss, actor_loss_grad,
100                                old_policy, params, maximal_step, args.max_kl,
101                                advantages, states, actions)
```

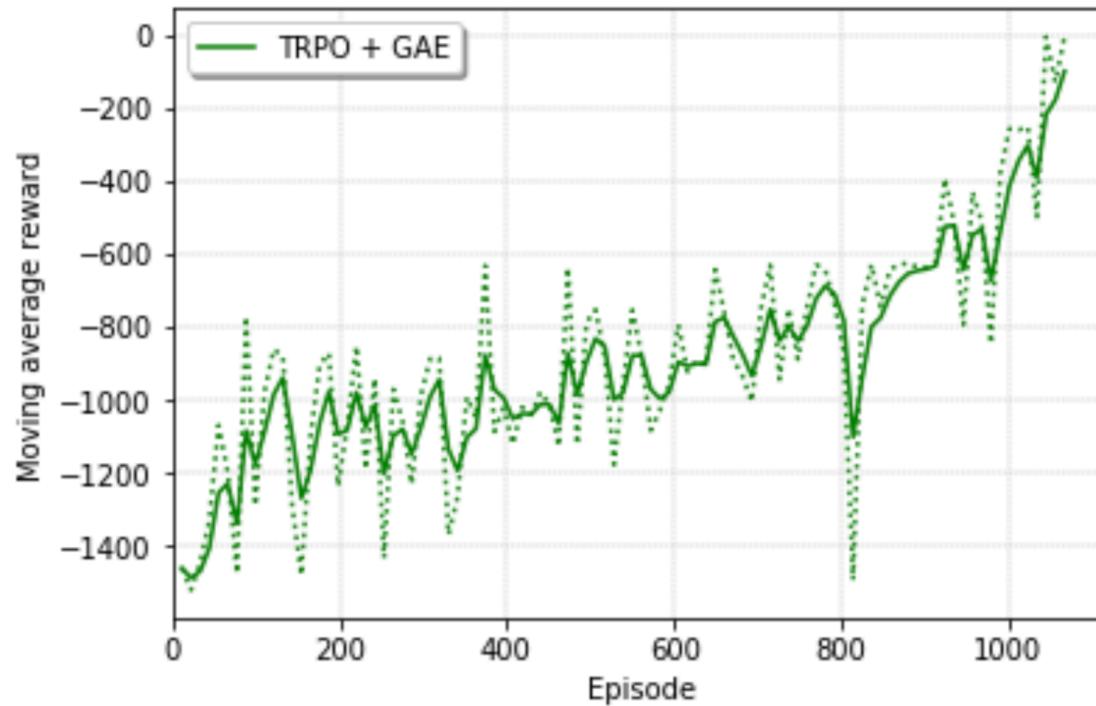
Train & TensorboardX

- 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

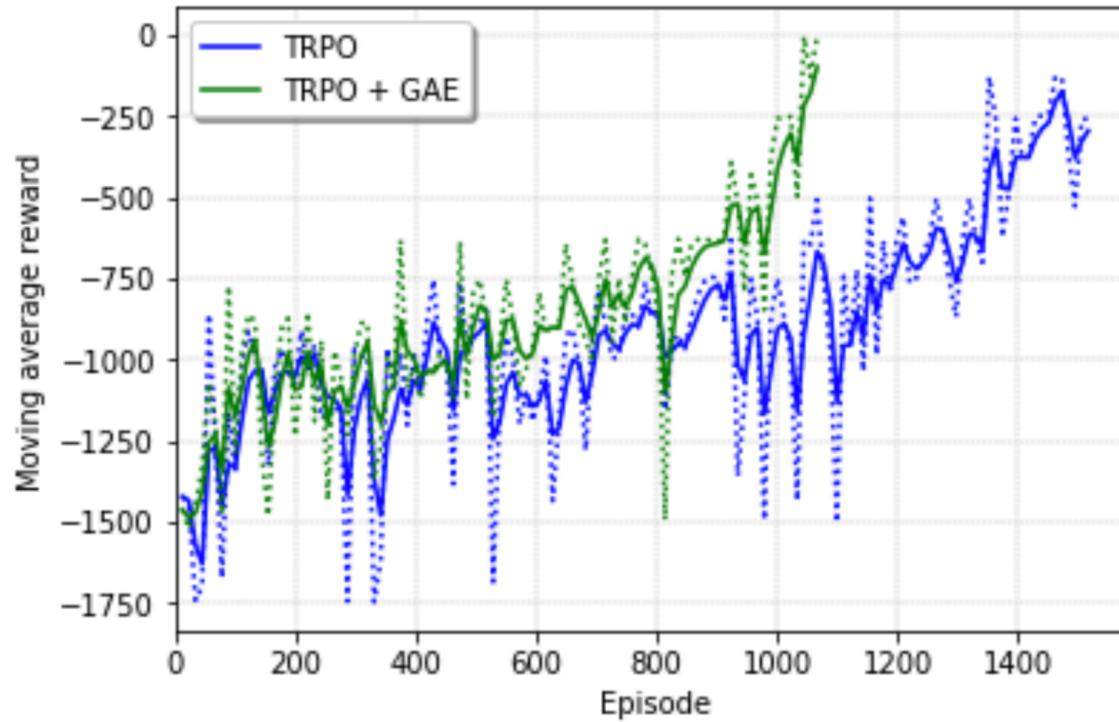
- Learning curve



- Test
 - `python test.py`

TRPO vs. TRPO + GAE

- Learning curve



Thank you



CORE
Control + Optimization Research Lab