CORE
Control + Optimization Research Lab

Generalized Advantage Estimator (GAE)

이동민

삼성전자 서울대 공동연구소

Jul 19, 2019

Outline

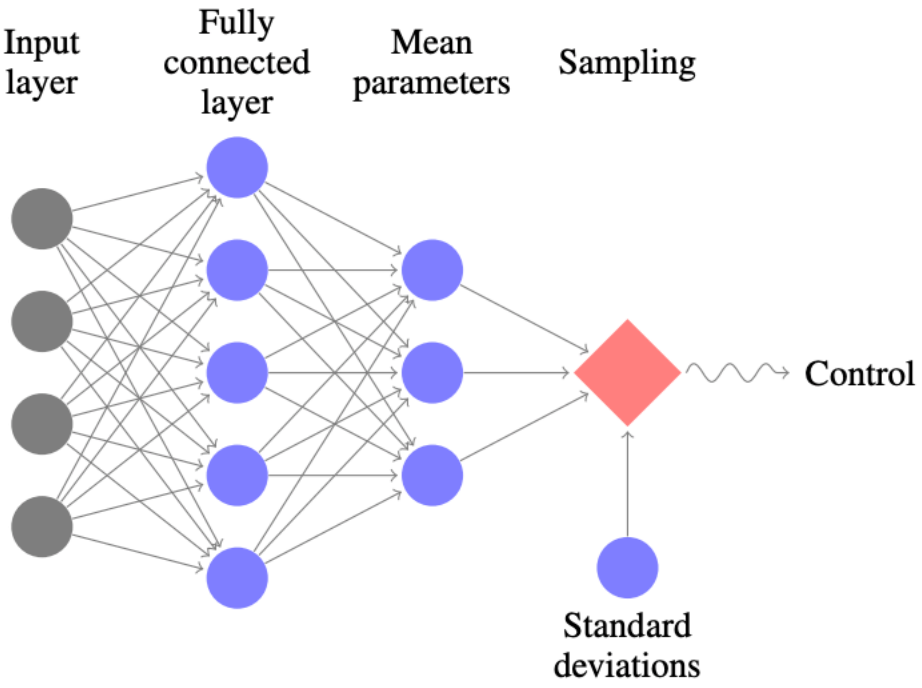
- Generalized Advantage Estimator (GAE)
  - Learning process
  - Hyperparameter
  - Main loop
  - Train model
  - Train & TensorboardX
  - Learning curve & Test
- TRPO vs. TRPO + GAE - Learning curve

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, s')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 : Actor loss의 gradient를 구하고, conjugate gradient method (CGM)을 통해 search direction 구하기
     - Step 4 : Step size와 maximal step 구하기
     - Step 5 : Actor network를 업데이트하고, backtracking line search를 통해 trust region을 잡고 trust region안에서 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

- Optimization problem of theoretical TRPO → Surrogate objective function

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

$$s.t. \ \bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old},\theta) \le \delta$$

- But how do we solve it? Solution → Approximation!

$$\mathcal{L}_{\theta_{old}}(\theta) \approx g^T(\theta-\theta_{old}) \qquad g \doteq \nabla_\theta \mathcal{L}_{\theta_{old}}(\theta)\,|_{\theta_{old}}$$

$$\bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old},\theta) \approx \frac{1}{2}(\theta-\theta_{old})^T H(\theta-\theta_{old}) \qquad H \doteq \nabla_\theta^2 \bar{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old},\theta)\,|_{\theta_{old}}$$

Learning process

❖ 정리

1) Find search direction and step size through CGM and Hessian of KL

- Search direction : $H^{-1}g$ (use CGM to solve $Hx = g$ for $x = H^{-1}g$)
- Step size : $\sqrt{\frac{2\delta}{g^T H^{-1} g}}$

2) 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 1 : Return과 GAE 구하기
     - time step 5까지 진행하고 episode가 끝났을 경우를 가정

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

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

$$G_3 = R_4 + \gamma R_5$$

$$G_4 = R_5$$

     - 거꾸로 계산하며 계산해놓은 return값을 이용

$$G_4 = R_5$$

$$G_3 = R_4 + \gamma G_4$$

$$G_2 = R_3 + \gamma G_3$$

$$G_1 = R_2 + \gamma G_2$$

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
   - Step 1 : Return과 GAE 구하기
     - Generalized advantage estimator (GAE)

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

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

     - 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

$\delta_5^V = r_5 - V^\pi(s_5)$	δ_5^V
$\delta_4^V = r_4 + \gamma V^\pi(s_5) - V^\pi(s_4)$	$\delta_4^V + \gamma\lambda\delta_5^V$
$\delta_3^V = r_3 + \gamma V^\pi(s_4) - V^\pi(s_3)$	$\delta_3^V + \gamma\lambda\delta_4^V + \gamma^2\lambda^2\delta_5^V$
$\delta_2^V = r_2 + \gamma V^\pi(s_3) - V^\pi(s_2)$	$\delta_2^V + \gamma\lambda\delta_3^V + \gamma^2\lambda^2\delta_4^V + \gamma^3\lambda^3\delta_5^V$
$\delta_1^V = r_1 + \gamma V^\pi(s_2) - V^\pi(s_1)$	$\delta_1^V + \gamma\lambda\delta_2^V + \gamma^2\lambda^2\delta_3^V + \gamma^3\lambda^3\delta_4^V + \gamma^4\lambda^4\delta_5^V$

Learning process

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

```
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 : Critic network 업데이트
     - Critic Loss
       - $J_V(\theta) = (V_\theta(s) - R)^2$

         Prediction Target
       - n = 2200
       - 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
```

```python
# ----------------------------
# step 2: update critic
criterion = torch.nn.MSELoss()

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

for _ in range(5):
    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]
        values_samples = critic(states_samples)

        target_samples = returns.unsqueeze(1)[mini_batch_index]

        critic_loss = criterion(values_samples, target_samples)
        critic_optimizer.zero_grad()
        critic_loss.backward()
        critic_optimizer.step()
```

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
   - Step 3 : Actor loss의 gradient를 구하고, CGM을 통해 search direction 구하기

```
# ----------------------------
# step 3: get gradient of actor loss and search direction through conjugate gradient method
mu, std = actor(torch.Tensor(states))
old_policy = get_log_prob(actions, mu, std)
actor_loss = surrogate_loss(actor, advantages, states, old_policy.detach(), actions)

actor_loss_grad = torch.autograd.grad(actor_loss, actor.parameters())
actor_loss_grad = flat_grad(actor_loss_grad)

search_dir = conjugate_gradient(actor, states, actor_loss_grad.data, nsteps=10)

actor_loss = actor_loss.data.numpy()
```

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
   - Step 4 : Step size와 maximal step 구하기

```
# -----------------------------
# step 4: get step size and maximal step
gHg = (hessian_vector_product(actor, states, search_dir) * search_dir).sum(0, keepdim=True)
step_size = torch.sqrt(2 * args.max_kl / gHg)[0]
maximal_step = step_size * search_dir
```

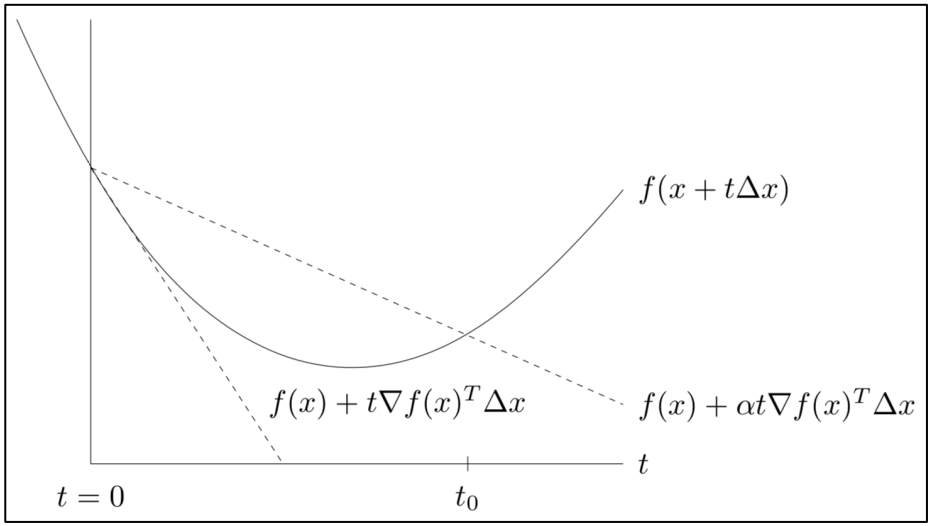
   - Step size : $\sqrt{\frac{2\delta}{g^T H^{-1} g}}$ $(Hx = g)$
   - Maximal step : $\sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$ (max_kl δ : 0.01)

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
   - Step 5 : Actor network를 업데이트하고, backtracking line search를 통해 trust region을 잡고 trust region안에서 업데이트

Algorithm 9.2 *Backtracking line search.*

given a descent direction Δx for f at $x \in \mathbf{dom}\, f$, $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$.
$t := 1$.
while $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x$, $\quad t := \beta t$.



source : https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

Learning process

4. 일정 sample들이 모이면 trajectories set으로 Actor & Critic network 업데이트
   - Step 5 : Actor network를 업데이트하고, backtracking line search를 통해 trust region을 잡고 trust region안에서 업데이트

```
backtrac_coef = 1.0
alpha = 0.5
beta = 0.5
flag = False

for i in range(10):
    new_params = params + backtrac_coef * maximal_step
    update_model(actor, new_params)

    new_actor_loss = surrogate_loss(actor, returns, states, old_policy.detach(), actions)
    new_actor_loss = new_actor_loss.data.numpy()

    loss_improve = new_actor_loss - actor_loss
    expected_improve *= backtrac_coef
    improve_condition = loss_improve / expected_improve

    kl = kl_divergence(new_actor=actor, old_actor=old_actor, states=states)
    kl = kl.mean()

    if kl < args.max_kl and improve_condition > alpha:
        flag = True
        break

    backtrac_coef *= beta

if not flag:
    params = flat_params(old_actor)
    update_model(actor, params)
    print('policy update does not impove the surrogate')
```

Hyperparameter

```
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('--critic_lr', type=float, default=1e-3)
parser.add_argument('--max_kl', type=float, default=1e-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
  - Critic Optimizer
  - TensorboardX
  - Recent rewards

```
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)
    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, 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)

```
def train_model(actor, critic, 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

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

```
# ----------------------------
# step 1: get returns and GAEs
values = critic(torch.Tensor(states))
returns, advantages = get_gae(rewards, masks, values, args)

# ----------------------------
# step 2: update critic
criterion = torch.nn.MSELoss()

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

for _ in range(5):
    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]
        values_samples = critic(states_samples)

        target_samples = returns.unsqueeze(1)[mini_batch_index]

        critic_loss = criterion(values_samples, target_samples)
        critic_optimizer.zero_grad()
        critic_loss.backward()
        critic_optimizer.step()
```

Train model

```
# ----------------------------
# step 3: get gradient of actor loss and search direction through conjugate gradient method
mu, std = actor(torch.Tensor(states))
old_policy = get_log_prob(actions, mu, std)
actor_loss = surrogate_loss(actor, advantages, states, old_policy.detach(), actions)

actor_loss_grad = torch.autograd.grad(actor_loss, actor.parameters())
actor_loss_grad = flat_grad(actor_loss_grad)

search_dir = conjugate_gradient(actor, states, actor_loss_grad.data, nsteps=10)

actor_loss = actor_loss.data.numpy()

# ----------------------------
# step 4: get step size and maximal step
gHg = (hessian_vector_product(actor, states, search_dir) * search_dir).sum(0, keepdim=True)
step_size = torch.sqrt(2 * args.max_kl / gHg)[0]
maximal_step = step_size * search_dir
```

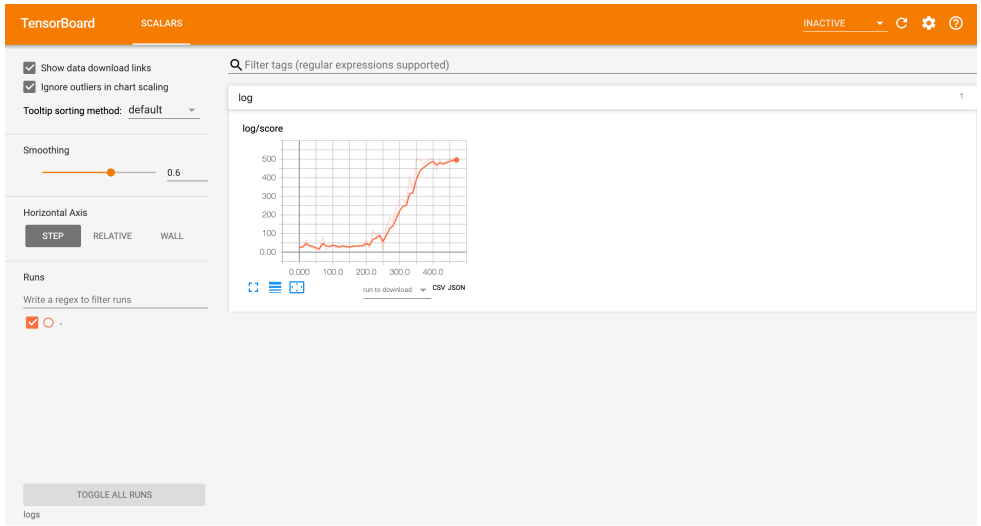
Train model

```
# ----------------------------
# step 5: update actor and perform backtracking line search for n iteration
params = flat_params(actor)

old_actor = Actor(state_size, action_size, args)
update_model(old_actor, params)

expected_improve = (actor_loss_grad * maximal_step).sum(0, keepdim=True)
expected_improve = expected_improve.data.numpy()

backtrac_coef = 1.0
alpha = 0.5
beta = 0.5
flag = False

for i in range(10):
    new_params = params + backtrac_coef * maximal_step
    update_model(actor, new_params)

    new_actor_loss = surrogate_loss(actor, advantages, states, old_policy.detach(), actions)
    new_actor_loss = new_actor_loss.data.numpy()

    loss_improve = new_actor_loss - actor_loss
    expected_improve *= backtrac_coef
    improve_condition = loss_improve / expected_improve

    kl = kl_divergence(new_actor=actor, old_actor=old_actor, states=states)
    kl = kl.mean()

    if kl < args.max_kl and improve_condition > alpha:
        flag = True
        break

    backtrac_coef *= beta
```

Train & TensorboardX

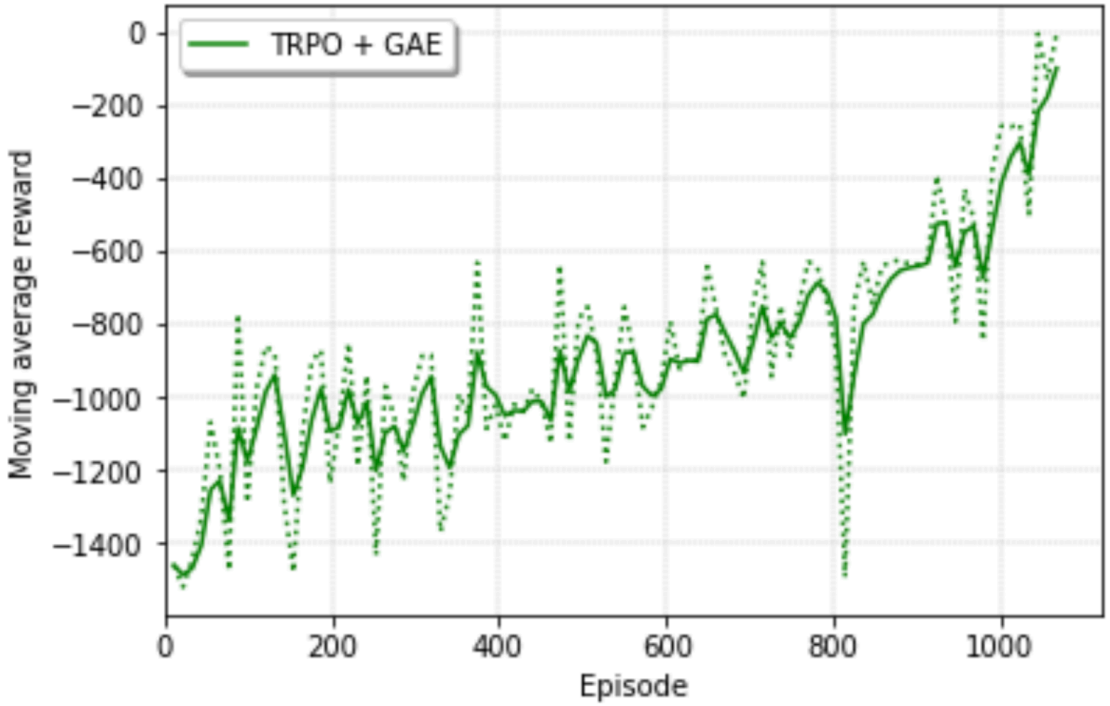
- Terminal A - train 실행
  - conda activate env_name
  - python train.py

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



Learning curve & Test

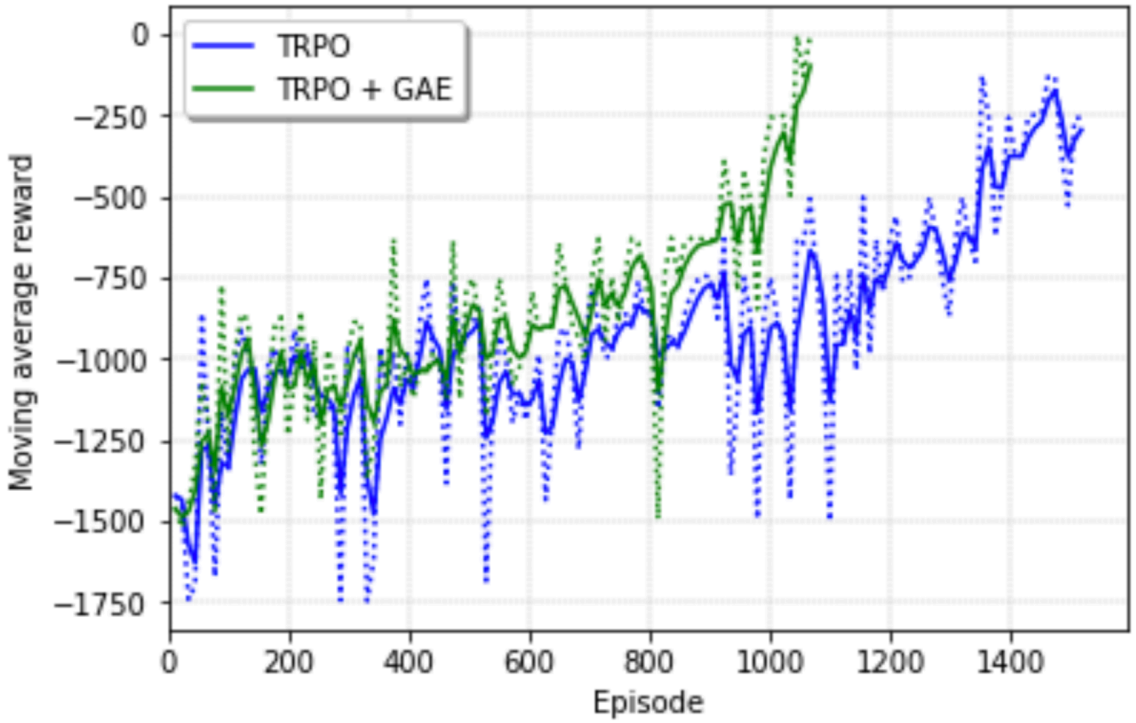
- Learning curve



- Test
  - ➢ python test.py

TRPO vs. TRPO + GAE

- Learning curve



Thank you