



**CORE**

Control + Optimization Research Lab

# Deep Deterministic Policy Gradient (DDPG)

이동민

삼성전자 서울대 공동연구소

Jul 17, 2019

# Outline

---

- Environment : Pendulum
- Deep Deterministic Policy Gradient (DDPG)
  - Learning process
  - Hyperparameter
  - Main loop
  - Train model
  - Train & TensorboardX
  - Learning curve & Test
  - DDPG video

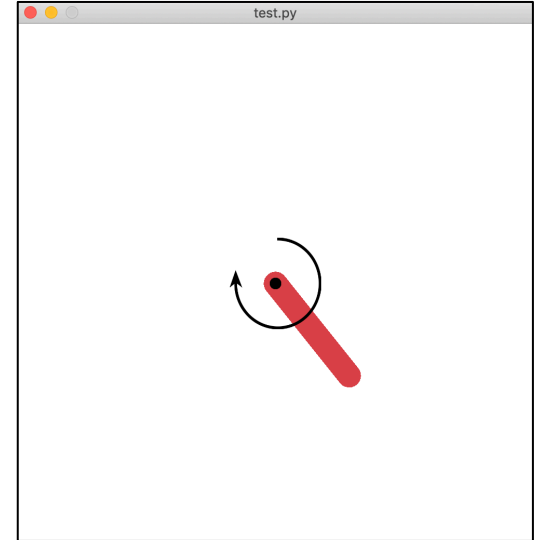
# Pendulum

- Env name : Pendulum-v0
- States : Continuous observation spaces

Num	Observation	Min	Max
0	$\cos(\theta)$	-1.0	1.0
1	$\sin(\theta)$	-1.0	1.0
2	$\dot{\theta}$	-8.0	8.0

- Actions : **Continuous** action spaces

Num	Action	Min	Max
0	Joint effort	-2.0	2.0



```
state: [0.88760426 0.46060687 0.60495138] | action: [1.5591747] |  
next_state: [0.8587901 0.5123276 1.18428273] | reward: -0.268161  
02634748273 | done: False  
state: [0.8587901 0.5123276 1.18428273] | action: [1.0582479] |  
next_state: [0.81139809 0.58449392 1.72726561] | reward: -0.430701  
2460303101 | done: False  
state: [0.81139809 0.58449392 1.72726561] | action: [0.7929939] |  
next_state: [0.73948894 0.6731687 2.28458514] | reward: -0.688669  
3000621688 | done: False  
state: [0.73948894 0.6731687 2.28458514] | action: [-0.6580073] |  
next_state: [0.64251266 0.76627507 2.69076057] | reward: -1.06772  
66789916096 | done: False  
state: [0.64251266 0.76627507 2.69076057] | action: [-1.4092577] |  
next_state: [0.51847703 0.85509156 3.05407822] | reward: -1.48817  
52666467798 | done: False
```



# Pendulum

- Reward

The precise equation for reward:

$$-(\theta^2 + 0.1 \cdot \dot{\theta}^2 + 0.001 \cdot a^2)$$

Theta is normalized between  $-\pi$  and  $\pi$ . Therefore, the lowest cost is  $-(\pi^2 + 0.1 \cdot 8^2 + 0.001 \cdot 2^2) = -16.2736044$ , and the highest cost is  $0$ . In essence, the goal is to remain at zero angle (vertical), with the least rotational velocity, and the least effort.

- Episode Termination

There is no specified termination. Adding a maximum number of steps might be a good idea.

NOTE: Your environment object could be wrapped by the TimeLimit wrapper, if created using the "gym.make" method. In that case it will terminate after 200 steps.

# Deep Deterministic Policy Gradient (DDPG)

- DDPG Algorithm

---

**Algorithm 1** DDPG algorithm

---

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .

Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer  $R$

**for** episode = 1, M **do**

    Initialize a random process  $\mathcal{N}$  for action exploration

    Receive initial observation state  $s_1$

**for** t = 1, T **do**

        Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise

        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$

        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$

        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$

        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

**end for**

**end for**

---

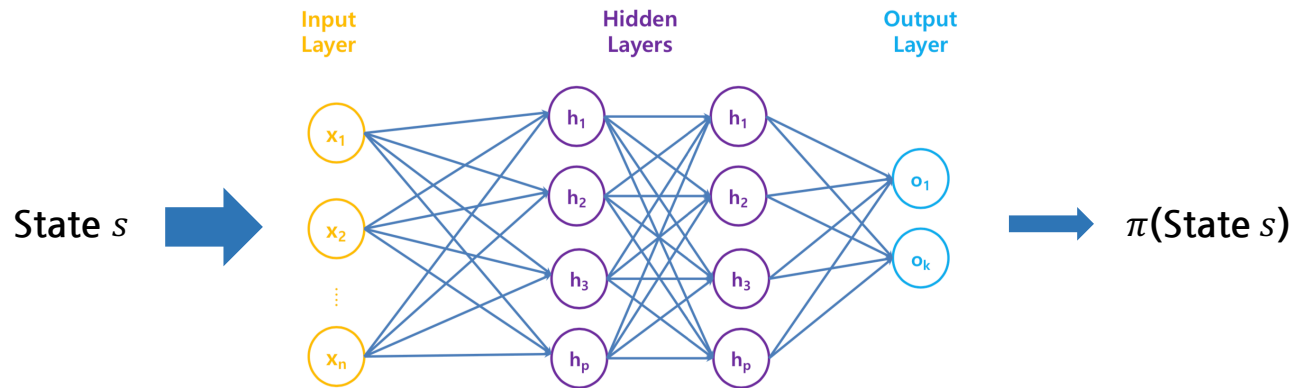
# Deep Deterministic Policy Gradient (DDPG)

- Learning process
  1. 상태에 따른 행동 선택
  2. 환경에서 선택한 행동으로 한 time step을 진행한 후, 다음 상태와 보상을 받음
  3. Sample  $(s, a, r, s')$ 을 replay buffer에 저장
  4. Replay buffer에서 랜덤으로 sample을 추출
  5. 추출한 sample로 Actor & Critic network 업데이트
  6. Actor & Critic에 대해 Soft target 업데이트

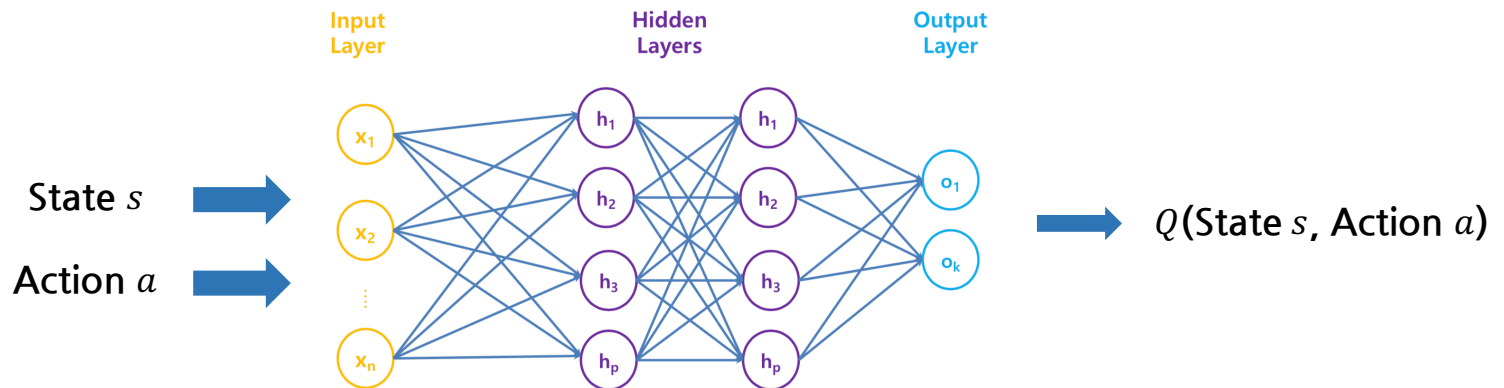


# Deep Deterministic Policy Gradient (DDPG)

- Actor network



- Critic network



# Deep Deterministic Policy Gradient (DDPG)

- 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.relu(self.fc1(x))
13         x = torch.relu(self.fc2(x))
14         policy = self.fc3(x)
15
16         return policy
```





# Deep Deterministic Policy Gradient (DDPG)

- Critic network

```
18 class Critic(nn.Module):
19     def __init__(self, state_size, action_size, args):
20         super(Critic, self).__init__()
21         self.fc1 = nn.Linear(state_size + action_size, args.hidden_size)
22         self.fc2 = nn.Linear(args.hidden_size, args.hidden_size)
23         self.fc3 = nn.Linear(args.hidden_size, 1)
24
25     def forward(self, states, actions):
26         x = torch.cat([states, actions], dim=1)
27         x = torch.relu(self.fc1(x))
28         x = torch.relu(self.fc2(x))
29         q_value = self.fc3(x)
30
31         return q_value
```



# Learning process

## 1. 상태에 따른 행동 선택

```
118 policy = actor(torch.Tensor(state))
119 action = get_action(policy, ou_noise, env) train.py
```

```
19 def get_action(policy, ou_noise, env):
20     action = policy.detach().numpy() + ou_noise.sample()
21     action = np.clip(action, env.action_space.low, env.action_space.high)
22
23     return action utils.py
```

- Ornstein-Uhlenbeck noise (OU noise) - theta : 0.15, mu : 0.0, sigma : 0.3

- $dx_t = \theta(\mu - x_t)dt + \sigma dW_t$

utils.py

```
4 class OUNoise:
5     def __init__(self, action_size, theta, mu, sigma):
6         self.action_size = action_size
7         self.theta = theta
8         self.mu = mu
9         self.sigma = sigma
10        self.X = np.zeros(self.action_size)
11
12    def sample(self):
13        dx = self.theta * (self.mu - self.X) + self.sigma * np.random.randn(len(self.X))
14        self.X = self.X + dx
15
16        return self.X
```

```
dx [-0.25447483]
X [-0.25447483]

dx [-0.20936269]
X [-0.46383752]

dx [0.1268112]
X [-0.33702632]

dx [0.18123875]
X [-0.15578757]

dx [-0.0240699]
X [-0.17985747]
```



# Learning process

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

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

3. Sample  $(s, a, r, s')$ 을 replay buffer에 저장

```
101         replay_buffer = deque(maxlen=100000)
```

```
124         mask = 0 if done else 1
```

```
125
```

```
126         replay_buffer.append((state, action, reward, next_state, mask))
```

4. Replay buffer에서 랜덤으로 sample을 추출 (Batch size : 128)

```
132         mini_batch = random.sample(replay_buffer, args.batch_size)
```



# Learning process

## 5. 추출한 sample로 Actor & Critic network 업데이트

- Critic Loss

$$J_Q(\phi) = ( \underbrace{r + \gamma Q_{\phi}(s', \pi_{\theta}(s'))}_{\text{Target}} - \underbrace{Q_{\phi}(s, a)}_{\text{Prediction}} )^2$$

```
51 # update critic
52 criterion = torch.nn.MSELoss()
53
54 # get Q-value
55 q_value = critic(torch.Tensor(states), actions).squeeze(1)
56
57 # get target
58 target_next_policy = target_actor(torch.Tensor(next_states))
59 target_next_q_value = target_critic(torch.Tensor(next_states), target_next_policy).squeeze(1)
60 target = rewards + masks * args.gamma * target_next_q_value
61
62 critic_loss = criterion(q_value, target.detach())
63 critic_optimizer.zero_grad()
64 critic_loss.backward()
65 torch.nn.utils.clip_grad_norm_(critic.parameters(), args.gradient_clip_critic)
66 critic_optimizer.step()
```



# Learning process

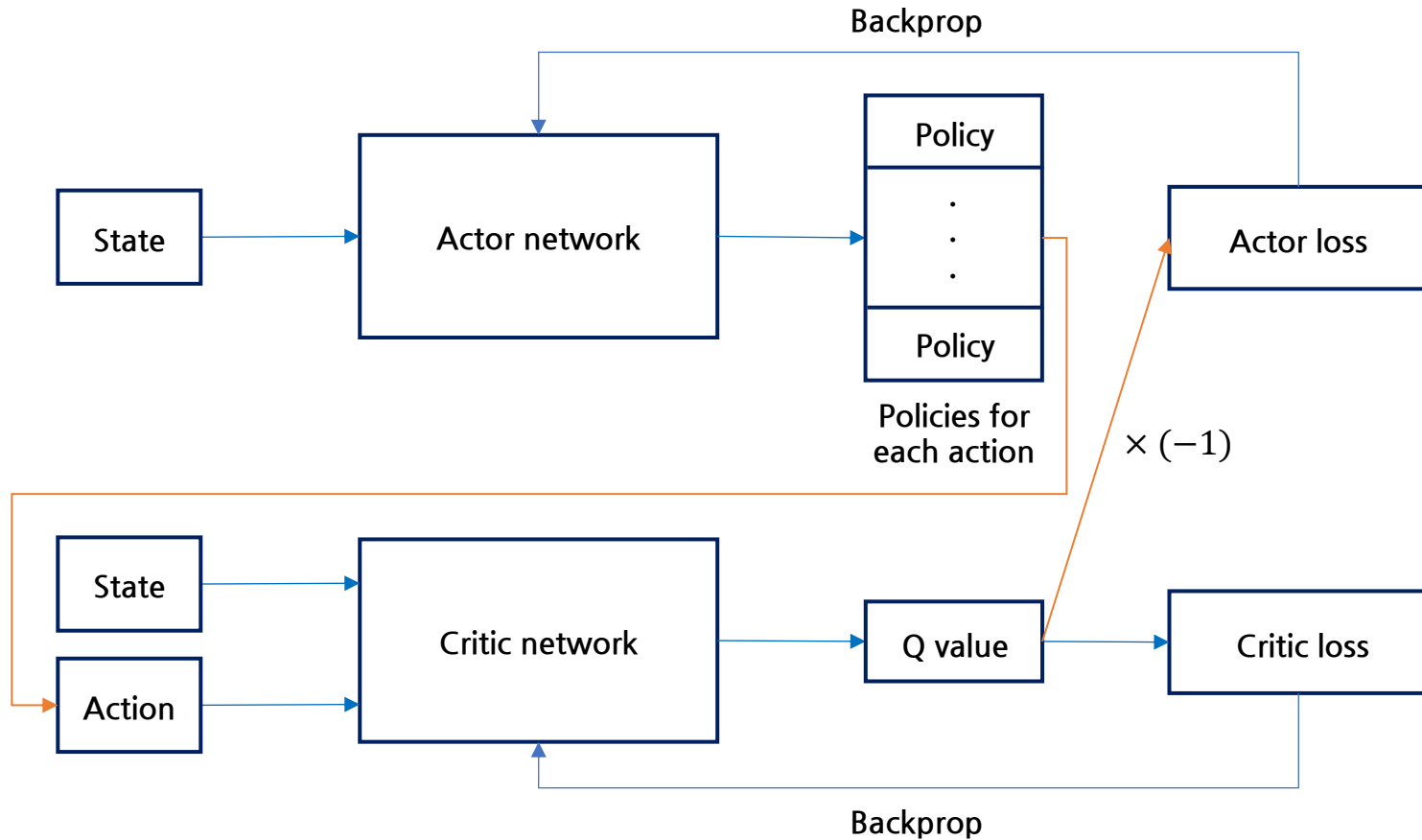
## 5. 추출한 sample로 Actor & Critic network 업데이트

- Actor Loss

$$J_{\pi}(\theta) = -\frac{1}{N} \sum Q_{\phi}(s, \pi_{\theta}(s))$$

```
68     # update actor
69     policy = actor(torch.Tensor(states))
70
71     actor_loss = -critic(torch.Tensor(states), policy).mean()
72     actor_optimizer.zero_grad()
73     actor_loss.backward()
74     torch.nn.utils.clip_grad_norm_(actor.parameters(), args.gradient_clip_actor)
75     actor_optimizer.step()
```

# Learning process



# Learning process

## 6. Actor & Critic에 대해 Soft target 업데이트

- Initialize target model

```
96 |         hard_target_update(actor, critic, target_actor, target_critic) train.py
24 | def hard_target_update(actor, critic, target_actor, target_critic):
25 |     target_critic.load_state_dict(critic.state_dict())
26 |     target_actor.load_state_dict(actor.state_dict()) utils.py
```

- Soft target update ( $\tau : 0.001$ )

$$\phi^{Q'} \leftarrow \tau \phi^Q + (1 - \tau) \phi^{Q'}$$

$$\theta^{\pi'} \leftarrow \tau \theta^{\pi} + (1 - \tau) \theta^{\pi'}$$

train.py

```
139 |         soft_target_update(actor, critic, target_actor, target_critic, args.tau)
28 | def soft_target_update(actor, critic, target_actor, target_critic, tau):
29 |     soft_update(critic, target_critic, tau)
30 |     soft_update(actor, target_actor, tau)
31 |
32 | def soft_update(net, target_net, tau):
33 |     for param, target_param in zip(net.parameters(), target_net.parameters()):
34 |         target_param.data.copy_(tau * param.data + (1.0 - tau) * target_param.data) utils.py
```



# Hyperparameter

```
15 parser = argparse.ArgumentParser()
16 parser.add_argument('--env_name', type=str, default="Pendulum-v0")
17 parser.add_argument('--load_model', type=str, default=None)
18 parser.add_argument('--save_path', default='./save_model/', help='')
19 parser.add_argument('--render', action="store_true", default=False)
20 parser.add_argument('--gamma', type=float, default=0.99)
21 parser.add_argument('--hidden_size', type=int, default=128)
22 parser.add_argument('--batch_size', type=int, default=128)
23 parser.add_argument('--actor_lr', type=float, default=1e-4)
24 parser.add_argument('--critic_lr', type=float, default=1e-3)
25 parser.add_argument('--theta', type=float, default=0.15)
26 parser.add_argument('--mu', type=float, default=0.0)
27 parser.add_argument('--sigma', type=float, default=0.3)
28 parser.add_argument('--tau', type=float, default=0.001)
29 parser.add_argument('--gradient_clip_actor', type=float, default=0.5)
30 parser.add_argument('--gradient_clip_critic', type=float, default=1.0)
31 parser.add_argument('--max_iter_num', type=int, default=1000)
32 parser.add_argument('--log_interval', type=int, default=10)
33 parser.add_argument('--goal_score', type=int, default=-300)
34 parser.add_argument('--logdir', type=str, default='./logs',
35 | | | | | help='tensorboardx logs directory')
36 args = parser.parse_args()
```





# Main loop

- Initialization
  - Seed - random number 고정
  - Actor & Critic network
  - Target actor & critic network
  - Actor & Critic optimizer
  - Hard target update
  - OU noise
  - TensorboardX
  - Replay buffer
  - Recent rewards

```
78 def main():
79     env = gym.make(args.env_name)
80     env.seed(500)
81     torch.manual_seed(500)
82
83     state_size = env.observation_space.shape[0]
84     action_size = env.action_space.shape[0]
85     print('state size:', state_size)
86     print('action size:', action_size)
87
88     actor = Actor(state_size, action_size, args)
89     target_actor = Actor(state_size, action_size, args)
90     critic = Critic(state_size, action_size, args)
91     target_critic = Critic(state_size, action_size, args)
92
93     actor_optimizer = optim.Adam(actor.parameters(), lr=args.actor_lr)
94     critic_optimizer = optim.Adam(critic.parameters(), lr=args.critic_lr)
95
96     hard_target_update(actor, critic, target_actor, target_critic)
97     ou_noise = OUNoise(action_size, args.theta, args.mu, args.sigma)
98
99     writer = SummaryWriter(args.logdir)
100
101     replay_buffer = deque(maxlen=100000)
102     recent_rewards = deque(maxlen=100)
103     steps = 0
```



# Main loop

- Episode 진행
  - 상태에 따른 행동 선택
  - 다음 상태와 보상을 받음
  - Replay buffer에 저장

```
105     for episode in range(args.max_iter_num):
106         done = False
107         score = 0
108
109         state = env.reset()
110         state = np.reshape(state, [1, state_size])
111
112         while not done:
113             if args.render:
114                 env.render()
115
116             steps += 1
117
118             policy = actor(torch.Tensor(state))
119             action = get_action(policy, ou_noise, env)
120
121             next_state, reward, done, _ = env.step(action)
122
123             next_state = np.reshape(next_state, [1, state_size])
124             mask = 0 if done else 1
125
126             replay_buffer.append((state, action, reward, next_state, mask))
127
128             state = next_state
129             score += reward
```

# Main loop

- Episode 진행
  - Replay buffer에서 랜덤으로 64개의 sample을 추출 → Mini batch
  - Train model
  - Soft target update

```
131         if steps > args.batch_size:
132             mini_batch = random.sample(replay_buffer, args.batch_size)
133
134             actor.train(), critic.train()
135             target_actor.train(), target_critic.train()
136             train_model(actor, critic, target_actor, target_critic,
137                         actor_optimizer, critic_optimizer, mini_batch)
138
139             soft_target_update(actor, critic, target_actor, target_critic, args.tau)
140
141         if done:
142             recent_rewards.append(score)
```

# Main loop

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

```
144         if episode % args.log_interval == 0:
145             print('{} episode | score_avg: {:.2f}'.format(episode, np.mean(recent_rewards)))
146             writer.add_scalar('log/score', float(score), episode)
147
148         if np.mean(recent_rewards) > args.goal_score:
149             if not os.path.isdir(args.save_path):
150                 os.makedirs(args.save_path)
151
152             ckpt_path = args.save_path + 'model.pth.tar'
153             torch.save(actor.state_dict(), ckpt_path)
154             print('Recent rewards exceed -300. So end')
155             break
```

# Train model

- Mini batch → Numpy array
- Mini batch에 있는 64개의 sample들을 각각 나눔
  - state - (64, 3)
  - action - (64, 1)
  - reward - (64)
  - next\_state - (64, 3)
  - mask - (64)

```
38 def train_model(actor, critic, target_actor, target_critic,
39                 actor_optimizer, critic_optimizer, mini_batch):
40     mini_batch = np.array(mini_batch)
41     states = np.vstack(mini_batch[:, 0])
42     actions = list(mini_batch[:, 1])
43     rewards = list(mini_batch[:, 2])
44     next_states = np.vstack(mini_batch[:, 3])
45     masks = list(mini_batch[:, 4])
46
47     actions = torch.Tensor(actions).squeeze(1)
48     rewards = torch.Tensor(rewards).squeeze(1)
49     masks = torch.Tensor(masks)
```



# Train model

- Prediction
  - q\_value - (64)
- Target
  - target\_next\_policy - (64, 1)
  - target\_next\_q\_value - (64)
  - target - (64)
- Update critic - MSE Loss
  - $J_Q(\phi) = (r + \gamma Q_{\phi}(s', \pi_{\theta}(s')) - Q_{\phi}(s, a))^2$

```
51     # update critic
52     criterion = torch.nn.MSELoss()
53
54     # get Q-value
55     q_value = critic(torch.Tensor(states), actions).squeeze(1)
56
57     # get target
58     target_next_policy = target_actor(torch.Tensor(next_states))
59     target_next_q_value = target_critic(torch.Tensor(next_states), target_next_policy).squeeze(1)
60     target = rewards + masks * args.gamma * target_next_q_value
61
62     critic_loss = criterion(q_value, target.detach())
63     critic_optimizer.zero_grad()
64     critic_loss.backward()
65     torch.nn.utils.clip_grad_norm_(critic.parameters(), args.gradient_clip_critic)
66     critic_optimizer.step()
```



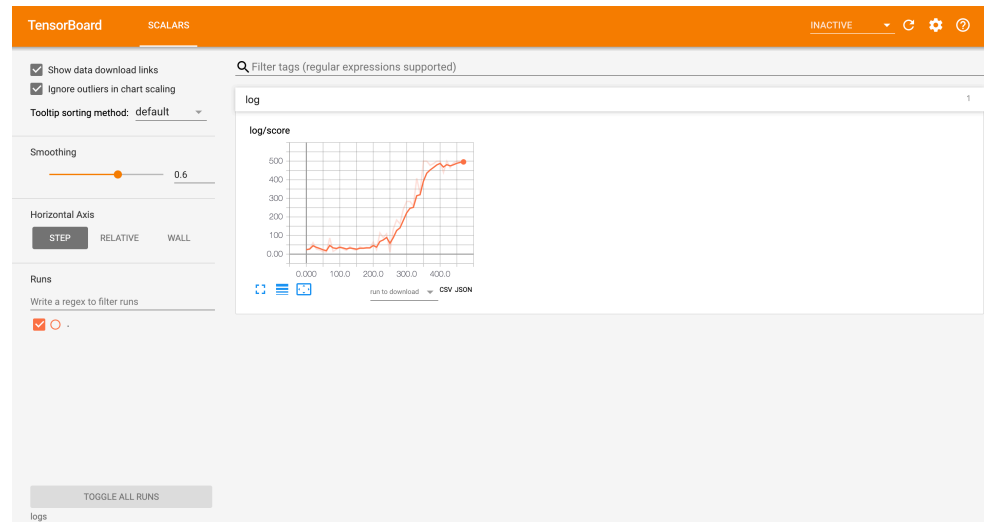
# Train model

- Update actor
  - policy - (64, 1)
  - critic(torch.Tensor(state), policy) - (64, 1)
  - $J_{\pi}(\theta) = -\frac{1}{N} \sum Q_{\phi}(s, \pi_{\theta}(s))$

```
68     # update actor
69     policy = actor(torch.Tensor(states))
70
71     actor_loss = -critic(torch.Tensor(states), policy).mean()
72     actor_optimizer.zero_grad()
73     actor_loss.backward()
74     torch.nn.utils.clip_grad_norm_(actor.parameters(), args.gradient_clip_actor)
75     actor_optimizer.step()
```

# Train & TensorboardX

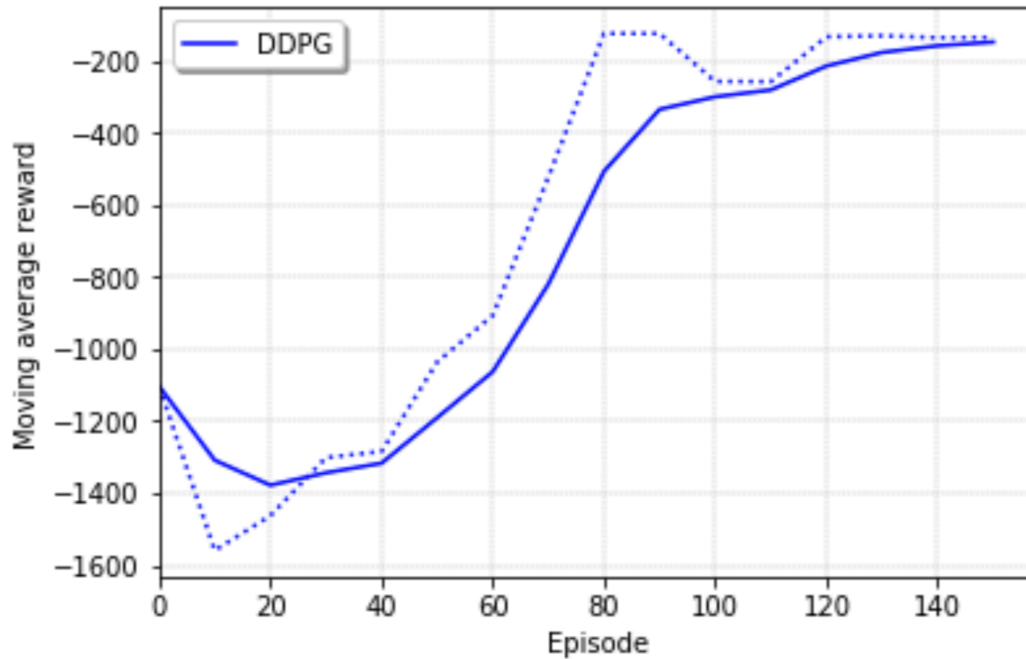
- Terminal A - train 실행
  - `conda activate env_name`
  - `python train.py`
- Terminal B - tensorboardX 실행
  - `conda activate env_name`
  - `tensorboard --logdir logs`
  - (웹에서) `localhost:6006`





# Learning curve & Test

- Learning curve



- Test
  - `python test.py`



Thank you

