Advantage Actor-Critic (A2C)

이동민

삼성전자 서울대 공동연구소
Jul 16, 2019

Outline

- Advantage Actor-Critic (A2C)
  - Learning process
  - Hyperparameter
  - Main loop
  - Train model
  - Train & TensorboardX
  - Learning curve & Test

- Comparison of algorithms for discrete action - Learning curve
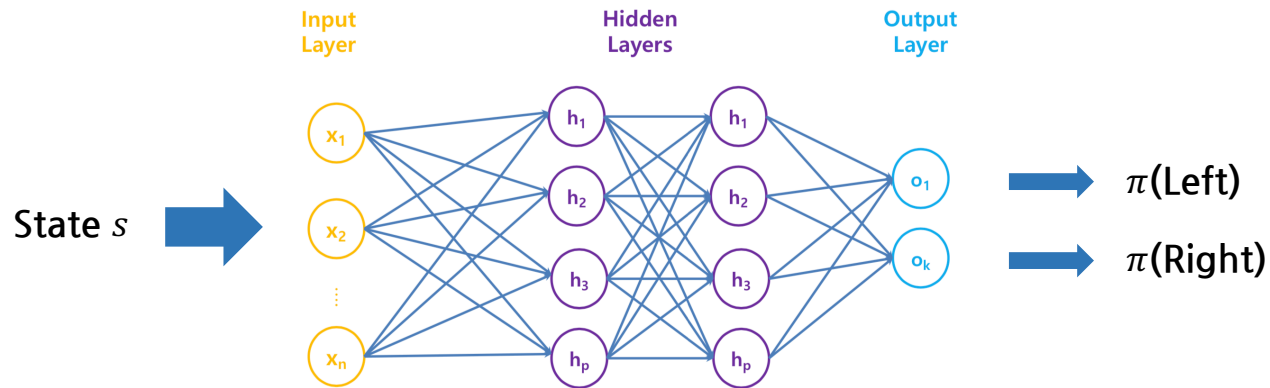
Advantage Actor-Critic (A2C)

- Learning process
    1. 상태에 따른 행동 선택
    2. 환경에서 선택한 행동으로 한 time step을 진행한 후, 다음 상태와 보상을 받음
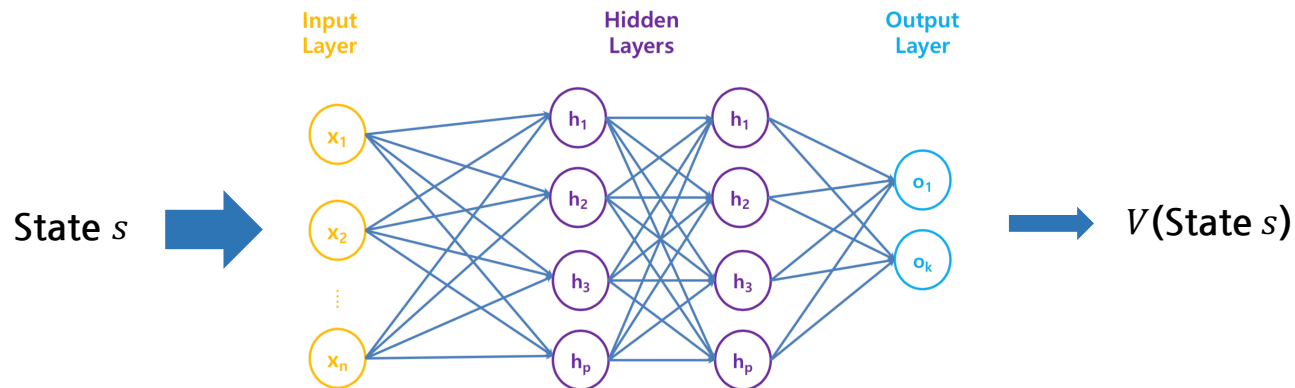   3. Sample (s, a, r, s')로 Actor & Critic network 업데이트

Advantage Actor-Critic (A2C)

- Actor network

State s → Input Layer (x_1, x_2, ..., x_n) → Hidden Layers (h_1, h_2, h_3, h_p) → Output Layer (o_1, o_k) → π(Left), π(Right)

- Critic network

State s → Input Layer (x_1, x_2, ..., x_n) → Hidden Layers (h_1, h_2, h_3, h_p) → Output Layer (o_1, o_k) → V(State s)

CORE
Control + Optimization Research Lab

Advantage Actor-Critic (A2C)

- Actor network

```python
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           policies = torch.softmax(self.fc3(x), dim=1)
15
16           return policies
```

CORE
Control + Optimization Research Lab

Advantage Actor-Critic (A2C)

- Critic network

```
18    class Critic(nn.Module):
19        def __init__(self, state_size, args):
20            super(Critic, self).__init__()
21            self.fc1 = nn.Linear(state_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, x):
26            x = torch.tanh(self.fc1(x))
27            x = torch.tanh(self.fc2(x))
28            value = self.fc3(x)
29
30            return value
```

Learning process

1. 상태에 따른 행동 선택

```
97              policies = actor(torch.Tensor(state))
98              action = get_action(policies)
```

```
58    def get_action(policies):
59        categorical = Categorical(policies)
60        action = categorical.sample()
61        action = action.data.numpy()[0]
62
63        return action
```

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

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

CORE
Control + Optimization Research Lab

Learning process

3. Sample (s, a, s', r)로 Actor & Critic network 업데이트
   - Critic Loss

$$J_V(\phi) = (\underline{r + \gamma V_\phi(s')} - \underline{V_\phi(s)})^2$$

Target   Prediction

```
33          # update critic
34          criterion = torch.nn.MSELoss()
35
36          value = critic(torch.Tensor(state)).squeeze(1)
37
38          next_value = critic(torch.Tensor(next_state)).squeeze(1)
39          target = reward + mask * args.gamma * next_value
40
41          critic_loss = criterion(value, target.detach())
42          critic_optimizer.zero_grad()
43          critic_loss.backward()
44          critic_optimizer.step()
```

CORE
Control + Optimization Research Lab

Learning process

3. Sample (s, a, s', r)로 Actor & Critic network 업데이트

   - **Actor Loss**

$$J_\pi(\theta) = -\log \pi_\theta(a|s) \underbrace{\left(r + \gamma V_\phi(s') - V_\phi(s) \right)}_{\textbf{Advantage Function}} + \underbrace{\alpha \left(-\sum_i \pi_{\theta_i} \log \pi_{\theta_i} \right)}_{\textbf{entropy}}$$

regularization term

   - Entropy is used to improve exploration by limiting the premature convergence to suboptimal policy.

Learning process

3. Sample (s, a, s', r)로 Actor & Critic network 업데이트

- Actor Loss

$$J_\pi(\theta) = -\log \pi_\theta(a|s) \left(r + \gamma V_\phi(s') - V_\phi(s) \right) + \alpha \left(-\sum_i \pi_{\theta_i} \log \pi_{\theta_i} \right)$$

```
46        # update actor
47        categorical = Categorical(policies)
48        log_policy = categorical.log_prob(torch.Tensor([action]))
49        entropy = categorical.entropy()
50
51        advantage = target - value
52
53        actor_loss = -log_policy * advantage.item() + args.ent_coef * entropy
54        actor_optimizer.zero_grad()
55        actor_loss.backward()
56        actor_optimizer.step()
```

CORE
Control + Optimization Research Lab

Learning process
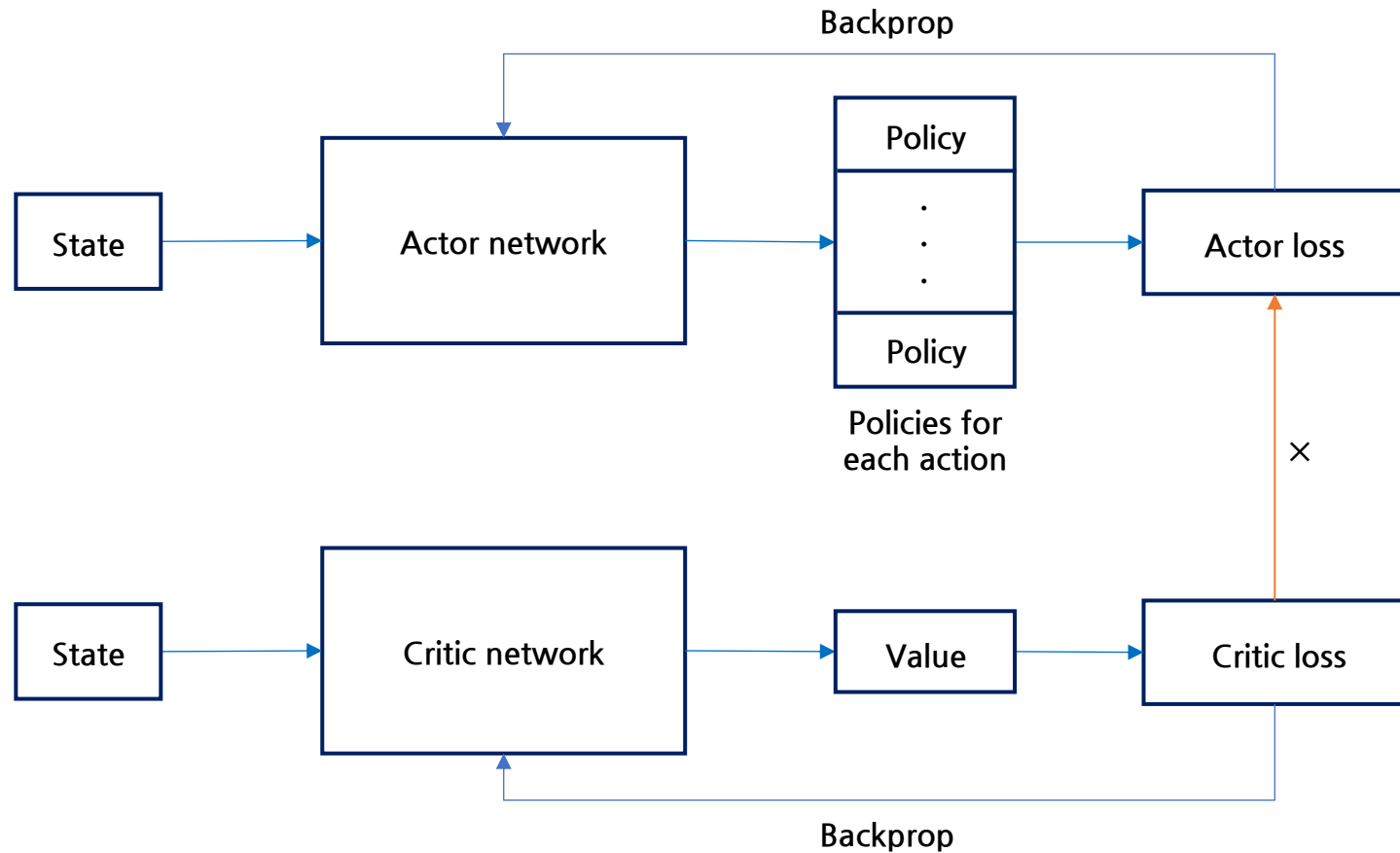
Hyperparameter

```python
13    parser = argparse.ArgumentParser()
14    parser.add_argument('--env_name', type=str, default="CartPole-v1")
15    parser.add_argument('--load_model', type=str, default=None)
16    parser.add_argument('--save_path', default='./save_model/', help='')
17    parser.add_argument('--render', action="store_true", default=False)
18    parser.add_argument('--gamma', type=float, default=0.99)
19    parser.add_argument('--hidden_size', type=int, default=64)
20    parser.add_argument('--actor_lr', type=float, default=1e-4)
21    parser.add_argument('--critic_lr', type=float, default=1e-3)
22    parser.add_argument('--ent_coef', type=float, default=0.1)
23    parser.add_argument('--max_iter_num', type=int, default=1000)
24    parser.add_argument('--log_interval', type=int, default=10)
25    parser.add_argument('--goal_score', type=int, default=400)
26    parser.add_argument('--logdir', type=str, default='./logs',
27                        help='tensorboardx logs directory')
28    args = parser.parse_args()
```

Main loop

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

```python
66    def main():
67        env = gym.make(args.env_name)
68        env.seed(500)
69        torch.manual_seed(500)
70
71        state_size = env.observation_space.shape[0]
72        action_size = env.action_space.n
73        print('state size:', state_size)
74        print('action size:', action_size)
75
76        actor = Actor(state_size, action_size, args)
77        critic = Critic(state_size, args)
78
79        actor_optimizer = optim.Adam(actor.parameters(), lr=args.actor_lr)
80        critic_optimizer = optim.Adam(critic.parameters(), lr=args.critic_lr)
81
82        writer = SummaryWriter(args.logdir)
83
84        running_score = 0
```

CORE
Control + Optimization Research Lab

Main loop

- Episode 진행
  - Reshape state vector (4) → $(1,4)$
  - 상태에 따른 행동 선택
  - 다음 상태와 보상을 받음
  - Reshape next state vector
  - Reward, mask 설정
  - Transition list에 저장
  - Train model
  - Running score 설정

```
86    for episode in range(args.max_iter_num):
87        done = False
88        score = 0
89
90        state = env.reset()
91        state = np.reshape(state, [1, state_size])
92
93        while not done:
94            if args.render:
95                env.render()
96
97            policies = actor(torch.Tensor(state))
98            action = get_action(policies)
99
100           next_state, reward, done, _ = env.step(action)
101
102           next_state = np.reshape(next_state, [1, state_size])
103           reward = reward if not done or score == 499 else -1
104           mask = 0 if done else 1
105
106           transition = [state, action, reward, next_state, mask]
107
108           actor.train(), critic.train()
109           train_model(actor, critic, actor_optimizer, critic_optimizer,
110                       transition, policies)
111
112           state = next_state
113           score += reward
114
115       score = score if score == 500.0 else score + 1
116       running_score = 0.99 * running_score + 0.01 * score
```

Main loop

- Print & Visualize log

- Running score > 400
  - Save model
  - 학습 종료

```
118         if episode % args.log_interval == 0:
119             print('{} episode | running_score: {:.2f}'.format(episode, running_score))
120             writer.add_scalar('log/score', float(score), episode)
121
122         if running_score > args.goal_score:
123             if not os.path.isdir(args.save_path):
124                 os.makedirs(args.save_path)
125
126             ckpt_path = args.save_path + 'model.pth.tar'
127             torch.save(actor.state_dict(), ckpt_path)
128             print('Running score exceeds 400. So end')
129             break
```

CORE
Control + Optimization Research Lab

Train model

- Transition List → state, action, next_state, reward, mask 각각 나누기

```
30    def train_model(actor, critic, actor_optimizer, critic_optimizer, transition, policies):
31        state, action, reward, next_state, mask = transition
```

- state - $(1, 4)$
- action
- reward
- next_state - $(1, 4)$
- mask

```
state [[-0.09066657 -1.56210361 -0.02062117  1.61977871]]
action 0
reward 1.0
next_state [[-0.12190864 -1.75697662  0.01177441  1.90596389]]
mask 1

state [[-0.12190864 -1.75697662  0.01177441  1.90596389]]
action 0
reward 1.0
next_state [[-0.15704817 -1.95222371  0.04989369  2.20227582]]
mask 1

state [[-0.15704817 -1.95222371  0.04989369  2.20227582]]
action 1
reward 1.0
next_state [[-0.19609265 -1.75761556  0.0939392   1.92538951]]
mask 1
```

CORE
Control + Optimization Research Lab

Train model

- Update critic - MSE Loss
  - $J_V(\phi) = (\underline{r + \gamma V_\phi(s')} - \underline{V_\phi(s)})^2$
    - Target   Prediction

```
33      # update critic
34      criterion = torch.nn.MSELoss()
35
36      value = critic(torch.Tensor(state)).squeeze(1)
37
38      next_value = critic(torch.Tensor(next_state)).squeeze(1)
39      target = reward + mask * args.gamma * next_value
40
41      critic_loss = criterion(value, target.detach())
42      critic_optimizer.zero_grad()
43      critic_loss.backward()
44      critic_optimizer.step()
```

CORE
Control + Optimization Research Lab

Train model

- Update actor

 $$J_\pi(\theta) = -\log \pi_\theta(a|s)\left(r + \gamma V_\phi(s') - V_\phi(s)\right) + \alpha\left(-\sum_i \pi_{\theta_i} \log \pi_{\theta_i}\right)$$

  Entropy coefficient : 0.1

```
46        # update actor
47        categorical = Categorical(policies)
48        log_policy = categorical.log_prob(torch.Tensor([action]))
49        entropy = categorical.entropy()
50
51        advantage = target - value
52
53        actor_loss = -log_policy * advantage.item() + args.ent_coef * entropy
54        actor_optimizer.zero_grad()
55        actor_loss.backward()
56        actor_optimizer.step()
```
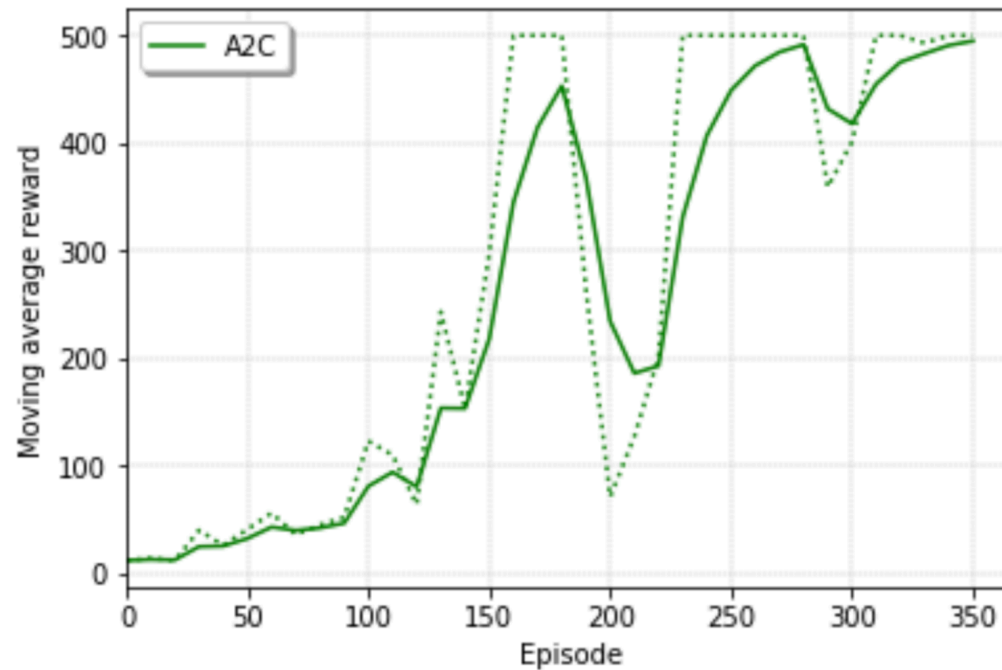
CORE
Control + Optimization Research Lab

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
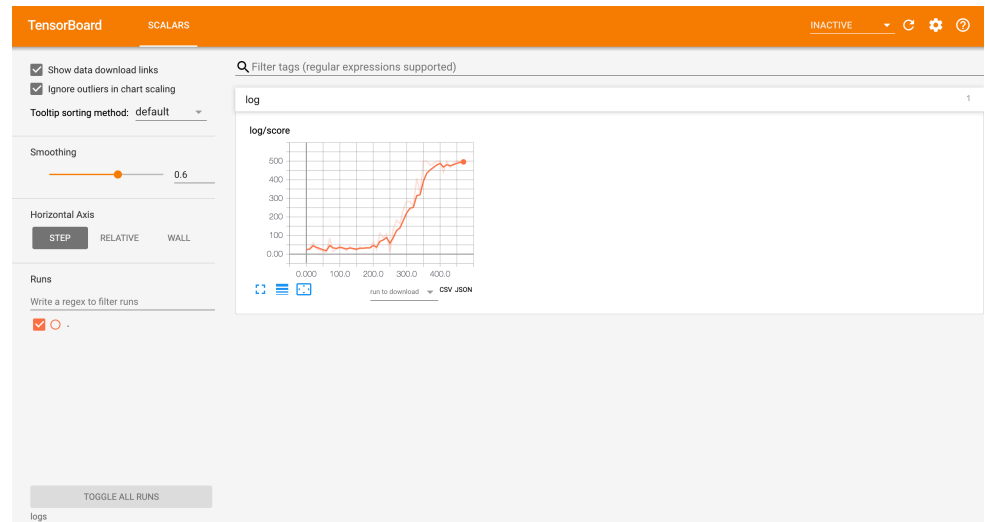
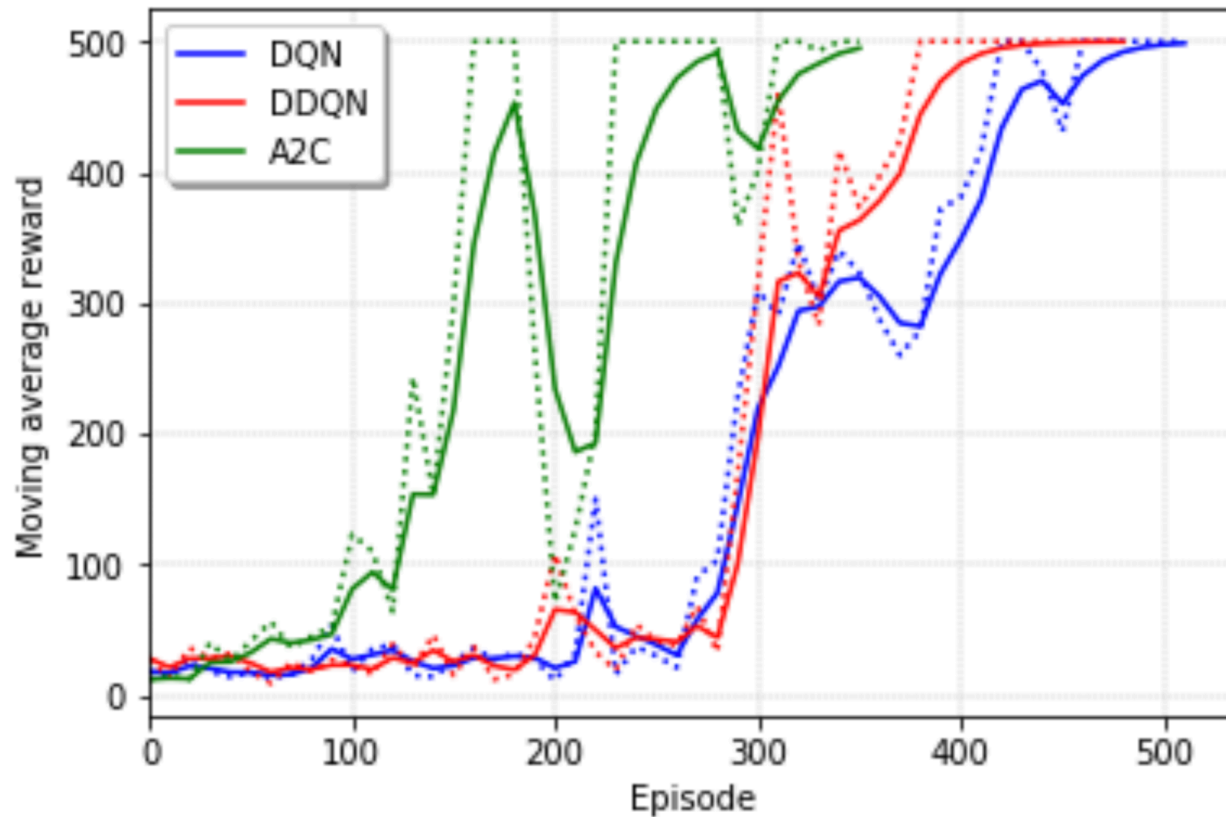# Learning curve & Test

- Learning curve

- Test
  - ➢ python test.py

Comparison of algorithms for discrete action

- Learning curve

Comparison of algorithms for discrete action

Discrete action일 때는 어떤 알고리즘이 제일 좋을까요?
그리고 그 알고리즘이 왜 좋을까요?

Comparison of algorithms for discrete action

Discrete action일 때는 어떤 알고리즘이 제일 좋을까요?
그리고 그 알고리즘이 왜 좋을까요?

- **DQN, DDQN (Off-policy algorithms)**
  - Replay buffer의 크기 → 컴퓨터의 메모리를 많이 차지하며 **느린 학습 속도**
  - Replay buffer를 통해 학습을 진행
    → 지금 policy가 아닌 **이전 policies**를 통해 모은 sample로 학습
  - ϵ-**greedy policy**를 통한 action 선택

- **A2C (On-policy algorithm)**
  - Replay buffer가 **필요하지 않음**
  - **현재 policy**를 통해 학습
  - Actor에서 나오는 **policy 자체가 action에 대한 확률(stochastic policy)**이므로 따로 action에 대한 exploration을 정해주지 않아도 됨
  - Actor와 Critic **나눠서 업데이트**

Thank you