Day 4. Deep Deterministic Policy Gradient

NPEX Reinforcement Learning

July 28, 2021

Jaeuk Shin, Minkyu Park

CORE
Control + Optimization Research Lab

DDPG - Review

Recap : DQN aims to learn Q, and choose action greedily as follows:

$$a_t = \arg\max_a Q(s_t, a; \theta),$$

Now, instead of choosing action a_t directly by solving

$$\max_a Q(s_t, a; \theta),$$

we employ a separate **actor network** π_ϕ and just select a_t by

$$a_t = \pi_\phi(s_t).$$

$\longrightarrow$ **Actor-Critic methods**

CORE
Control + Optimization Research Lab

DDPG - Review

Updating θ?

Training $Q(s, a; \theta)$ in DDPG is simple as same as training it in DQN:

$$\textbf{DQN}: \quad y_j = r_j + \gamma \max_a Q(s_j', a; \theta) \quad \textbf{(TD target)}$$

$$\textbf{DDPG}: \quad y_j = r_j + \gamma Q(s_j', \pi_\phi(s_j); \theta)$$

We solve (by performing gradient descent)

$$\min_\theta \frac{1}{|B|} \sum_j |Q(s_j, \boxed{a_j}; \theta) - y_j|^2.$$

not $\pi_\phi(s_j)$!

CORE
Control + Optimization Research Lab

DDPG - Review

How to tune params ϕ of the actor π_ϕ?

Our original goal was to solve

$$\max_a Q(s_t, a; \theta),$$

so we expect π_ϕ to satisfy

$$Q(s_t, \pi_\phi(s_t); \theta) \approx \max_a Q(s_t, a; \theta)$$

This naturally leads to the following optimization problem!

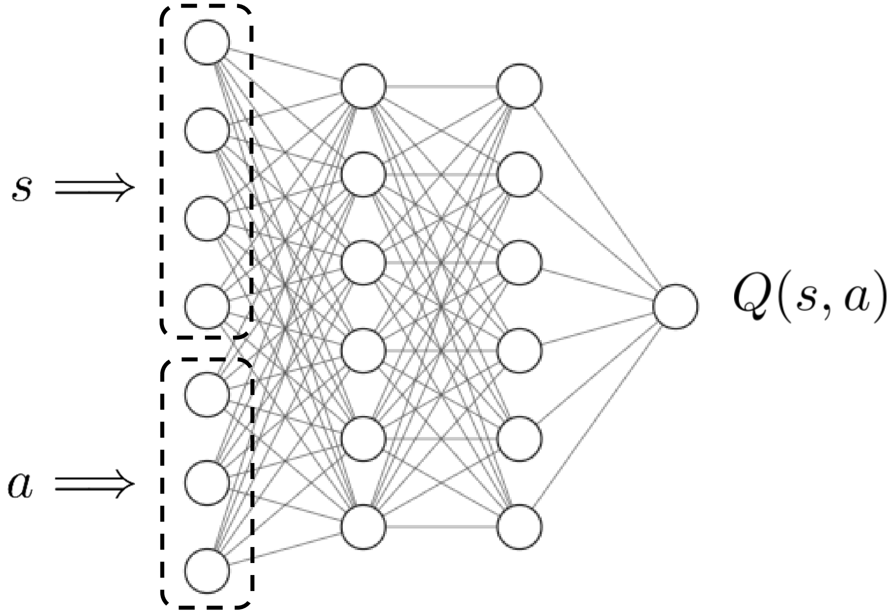
$$\max_\phi Q(s_t, \pi_\phi(s_t); \theta)$$

DDPG - Implementation

```
class Critic(nn.Module):
    def __init__(self, obs_dim, act_dim, hidden1, hidden2):
        super(Critic, self).__init__()
        self.fc1 = nn.Linear(obs_dim + act_dim, hidden1)
        self.fc2 = nn.Linear(hidden1, hidden2)
        self.fc3 = nn.Linear(hidden2, 1)

    def forward(self, obs, act):
        x = torch.cat([obs, act], dim=1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

feed state-action pair

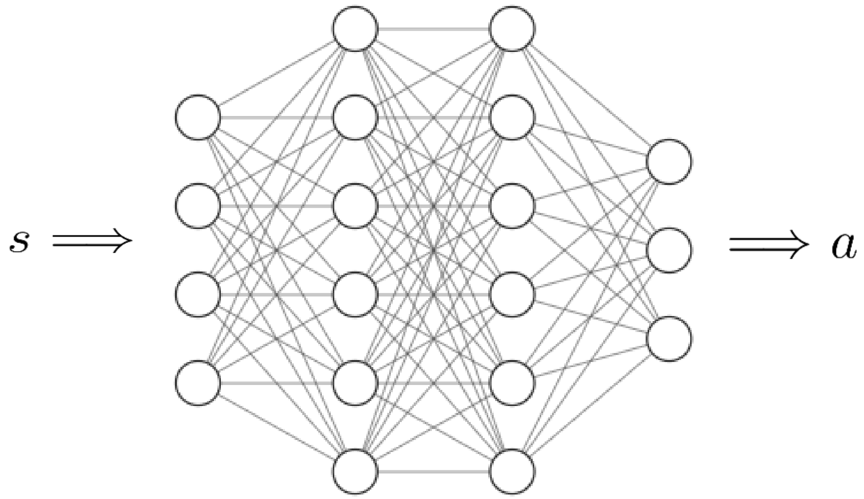


CORE
Control + Optimization Research Lab

DDPG - Implementation

```python
class Actor(nn.Module):
    def __init__(self, obs_dim, act_dim, ctrl_range, hidden1, hidden2):
        super(Actor, self).__init__()
        self.fc1 = nn.Linear(obs_dim, hidden1)
        self.fc2 = nn.Linear(hidden1, hidden2)
        self.fc3 = nn.Linear(hidden2, act_dim)
        self.ctrl_range = ctrl_range

    def forward(self, obs):
        x = F.relu(self.fc1(obs))
        x = F.relu(self.fc2(x))
        return self.ctrl_range * torch.tanh(self.fc3(x))
```



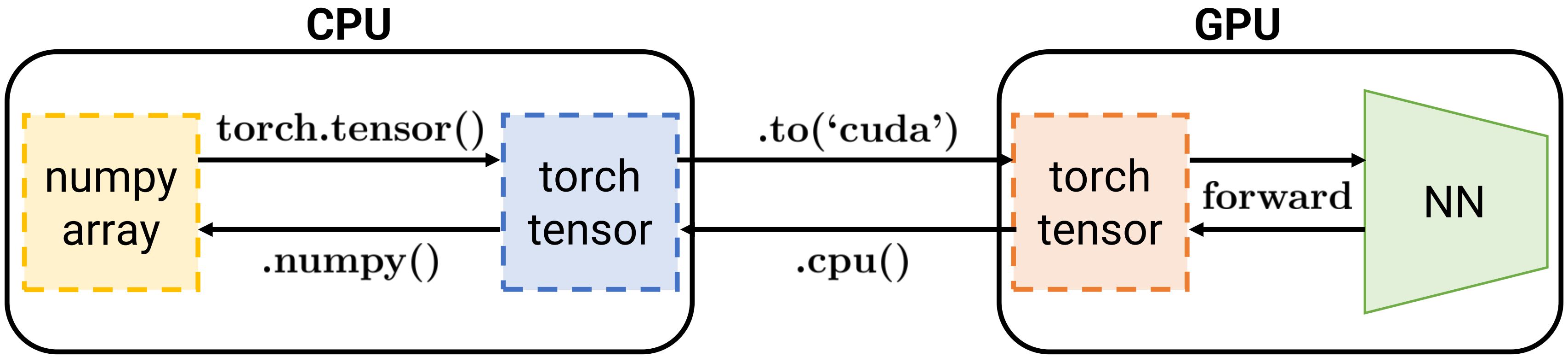
for control contraints $-\bar{u} \leq u \leq \bar{u}$ (physical limitation)

DDPG - Implementation

How to compute an action/control?

```python
def act(self, obs):
    obs = obs[np.newaxis, ...]
    with torch.no_grad():
        obs_tensor = torch.Tensor(obs).to(device)
        act_tensor = self.actor(obs_tensor)
    action = act_tensor.cpu().detach().numpy()
    action = np.squeeze(action, axis=0)
    return action
```

main part!



DDPG - Implementation

```
target = rewards + self.gamma * mask * self.targ_Q(next_observations, self.targ_pi(next_observations))
```

```
out = self.Q(observations, actions)
loss_ftn = MSELoss()
loss = loss_ftn(out, target)
self.Q_optimizer.zero_grad()
loss.backward()
self.Q_optimizer.step()
```

$Q(s, a; \theta)$

This is how we train DDPG!

```
pi_loss = - torch.mean(self.Q(observations, self.pi(observations)))
self.pi_optimizer.zero_grad()
pi_loss.backward()
self.pi_optimizer.step()
```

$\pi_\phi(s)$

Tip. freeze networks properly

CORE
Control + Optimization Research Lab

DDPG - Implementation

1. critic network training : what's different from DQN?

```python
def update(agent, replay_buf, gamma, actor_optim, critic_optim, target_actor, target_critic, tau, batch_size):

    batch = replay_buf.sample_batch(batch_size=batch_size)

    with torch.no_grad():
        obs = torch.Tensor(batch.obs).to(device)
        act = torch.Tensor(batch.act).to(device)
        next_obs = torch.Tensor(batch.next_obs).to(device)
        rew = torch.Tensor(batch.rew).to(device)
        done = torch.Tensor(batch.done).to(device)
        mask = 1. - done

        target = rew + gamma * mask * target_critic(next_obs, target_actor(next_obs))
    out = agent.critic(obs, act)

    loss_ftn = MSELoss()
    critic_loss = loss_ftn(out, target)
    critic_optim.zero_grad()
    critic_loss.backward()
    critic_optim.step()
```

$Q\left(s', \pi\left(s'\right)\right)$ instead of $\max_{a'} Q\left(s', a'\right)$!

DDPG - Implementation

2. actor network training

```
actor_loss = -torch.mean(agent.critic(obs, agent.actor(obs)))

actor_optim.zero_grad()
actor_loss.backward()
actor_optim.step()
```

This tries to solve
$\max_\phi Q(s_t, \pi_\phi(s_t); \theta)$.

DDPG - Implementation

Putting these together, we get...

```
obs = env.reset()
done = False
step_count = 0

for t in range(num_updates):
    action = agent.act(obs) + noise_std * np.random.randn(act_dim)
    action = np.clip(action, -ctrl_range, ctrl_range)

    next_obs, rew, done, _ = env.step(action)
    replay_buf.append(obs, action, next_obs, rew, done)
    obs = next_obs
    step_count += 1

    if step_count == ep_len:
        obs = env.reset()
        done = False
        step_count = 0

    if t % train_interval == 0:
        for _ in range(train_interval):
            update(...)
```

Why some random noise?

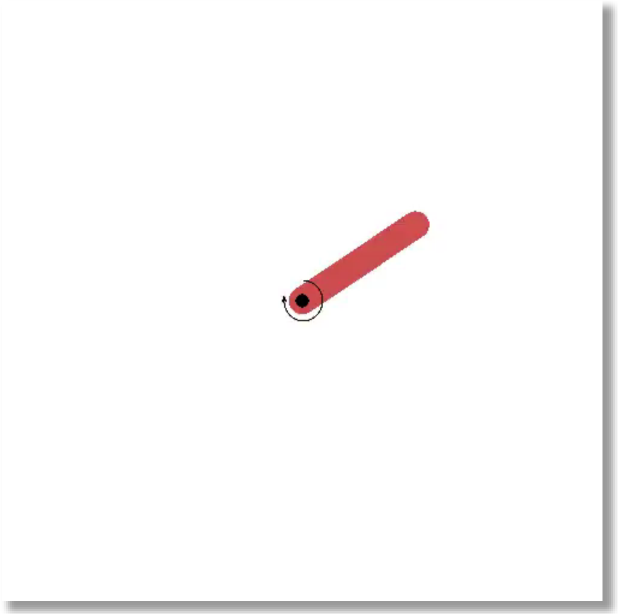
$\Longrightarrow$ For exploration! (cf. ε-greedy)

reset the episode

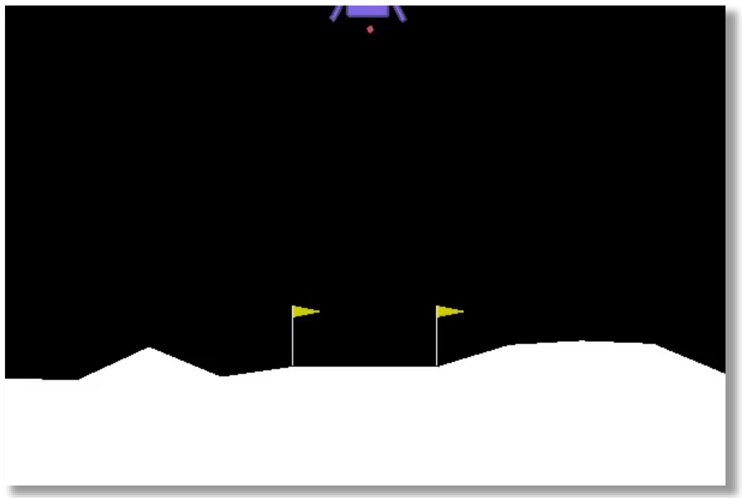
DDPG - Experiments

Task 1. Pendulum-v0 (see plot.py & test.py)

toy problem $\longrightarrow$



Task 2. LunarLanderContinuous-v2



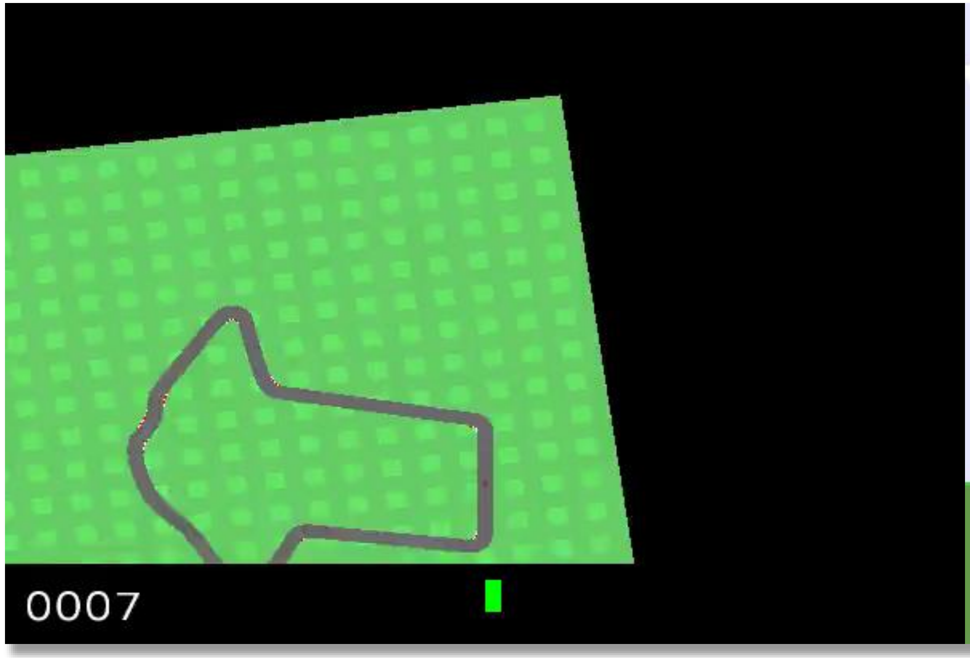
$\longleftarrow$ bit challenging!

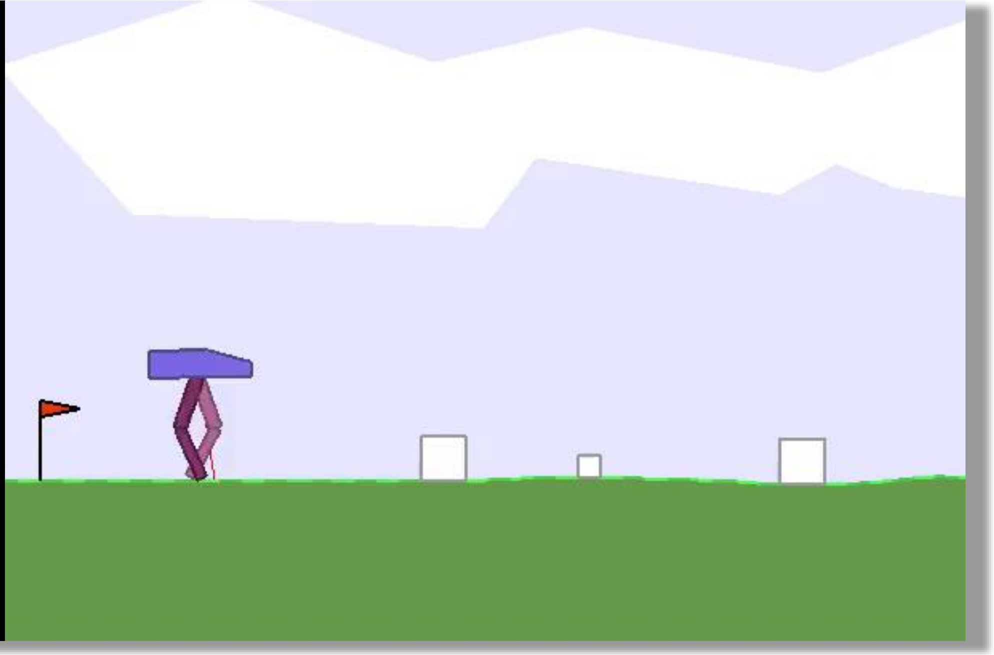
DDPG - Experiments

Install **Box2D** environments(including LunarLander) by

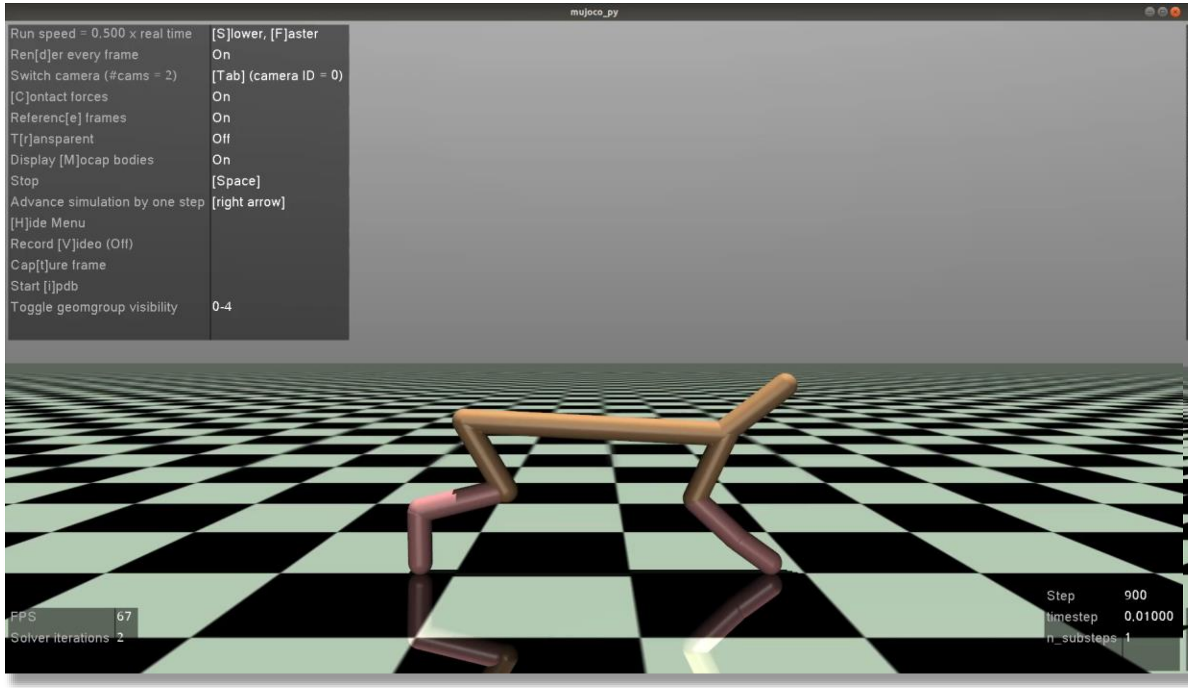
pip install Box2D

includes some challenging problems(try these with DDPG!)

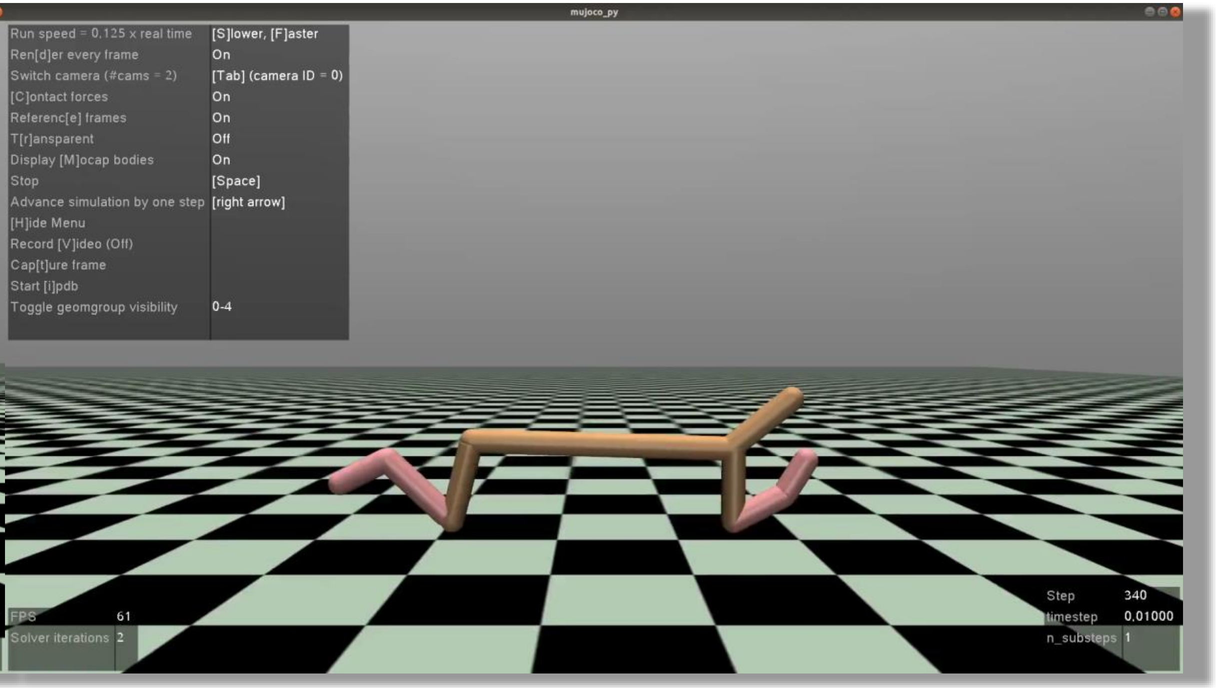




To try challenging & realistic problems, try **Pybullet**(free!) or **MuJoCo**.



untrained



trained

Thank you!