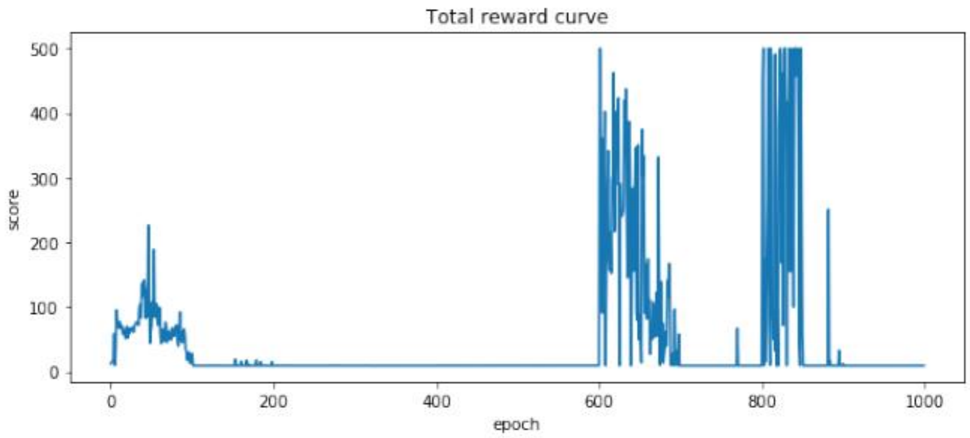
LAB 6: Deep Q-Network and Deep Deterministic Policy Gradient

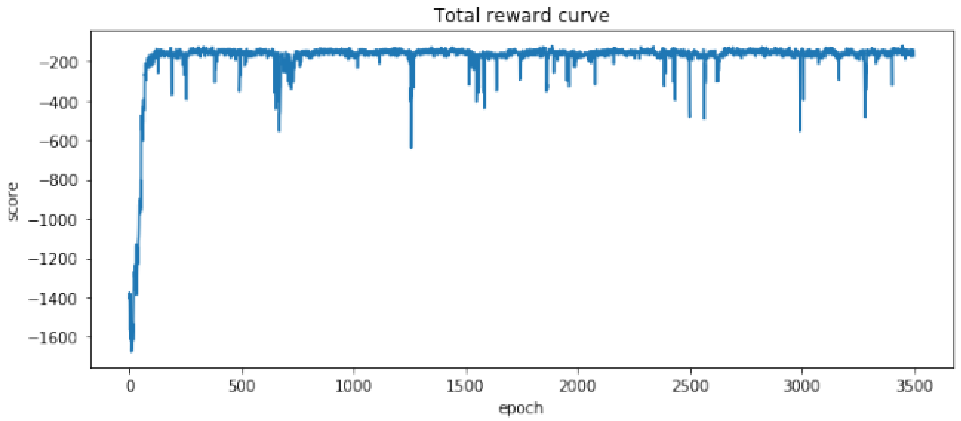
0851915 徐玉山

Report:

- A plot shows episode rewards of at least 1000 training episodes in CartPole-v1 (5%)



- A plot shows episode rewards of at least 1000 training episodes in Pendulum-v0 (5%)



- Describe your major implementation of both algorithms in detail. (20%)

```python
for episode in range(args.episode):
    total_reward = 0
    state = env.reset()
    for t in itertools.count(start=1):
        # select action
        if total_steps < args.warmup:
            action = env.action_space.sample()
        else:
            state_tensor = torch.Tensor(state).to(args.device)
            action = select_action(epsilon, state_tensor)
            epsilon = max(epsilon * args.eps_decay, args.eps_min)
        # execute action
        next_state, reward, done, _ = env.step(action)
        # store transition
        memory.append(state, [action], [reward / 10], next_state, [int(done)])
        if total_steps >= args.warmup and total_steps % args.freq == 0:
            # update the behavior network
            update_eval_network()
        if total_steps % args.target_freq == 0:
            # TODO: update the target network by copying from the behavior network
            target_net.load_state_dict(eval_net.state_dict())
            #raise NotImplementedError
```

```python
for episode in range(args.episode):
    total_reward = 0
    random_process.reset()
    state = env.reset()
    for t in itertools.count(start=1):
        # select action
        if total_steps < args.warmup:
            action = float(env.action_space.sample())
        else:
            state_tensor = torch.Tensor(state).to(args.device)
            action = select_action(state_tensor)
        # execute action
        next_state, reward, done, _ = env.step([action])
        # store transition
        memory.append(state, [action], [reward], next_state, [int(done)])
        if total_steps >= args.warmup:
            # update the behavior networks
            update_behavior_network()
            # update the target networks
            update_target_network(target_actor_net, actor_net)
            update_target_network(target_critic_net, critic_net)
```

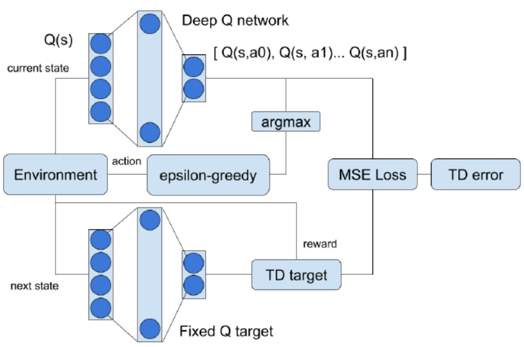
DQN DDPG

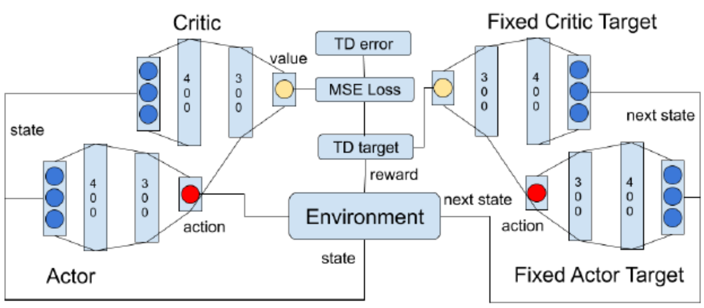
兩者的主要訓練架構挺相似的

1. 在steps < args.warmup 時都不會更新參數
2. 都會從環境中取得 next_state, reward, done 等資訊並存入memory裡面，並以此做 Experience replay
3. Train的時候都會從memory隨機挑選batchsize的數量來進行參數更新
4. DQN 具有target 和 eval 兩種網路，eval用來更新，target則會固定一定Steps數後才更新成eval的參數。

在每個epoch, 重設environment, 並把action送給環境(DQN使用 epsilon-greedy)
當environment回傳了next_state, reward, done等等資訊時
把這些資訊塞進buffer
當buffer量足夠時(step>args.warmup)再從buffer裡面隨機拿batchsize(DQN: 128 , DDPG:64) 數量的資料來更新網路。
當結束(done=1)的時候，把總reward以及loss等資訊存起來或是輸出

- Describe differences between your implementation and algorithms. (10%)





DQN DDPG

- Describe your implementation and the gradient of actor updating. (10%)

$$L = -Q(s, a|\theta_Q),\ a = u(s|\theta_u)$$

$$\frac{\nabla L}{\nabla \theta_u} = -\frac{\nabla Q(s,a|\theta_Q)}{\nabla a}\frac{\nabla a}{\nabla u(s|\theta_u)}\frac{\nabla u(s|\theta_u)}{\nabla \theta_u}$$
$$= -\frac{\nabla Q(s,a|\theta_Q)}{\nabla u(s|\theta_u)}\frac{\nabla u(s|\theta_u)}{\nabla \theta_u}$$

```
actor_loss = -critic_net(state , actor_net(state))
actor_loss = actor_loss.mean()
## update actor ##
# TODO: actor loss
# action = ?
# actor_loss = ?
#raise NotImplementedError
# optimize actor
actor_net.zero_grad()
critic_net.zero_grad()
actor_loss.backward()
actor_opt.step()
```

看似計算複雜但是這些都不用煩惱，pytorch的 autograd 直接可以計算

$$\theta_{\hat{Q}} = (1-\tau)\theta_{\hat{Q}} + \tau\theta_Q \theta_{\hat{u}} = (1-\tau)\theta_{\hat{u}} + \tau\theta_u$$

間隔一段時間之後再把target更新, tau 是

- Describe your implementation and the gradient of critic updating. (10%)

```
with torch.no_grad():
    next_q_values = target_critic_net( state_next, target_actor_net(state_next))
    next_q_values[np.argwhere(done == 1).reshape(-1)] = 0.0
    target_q = reward + args.gamma * next_q_values

## update critic ##
critic_net.zero_grad()
q = critic_net(state , action)
criterion = nn.MSELoss()
critic_loss = criterion(q, target_q)
# optimize critic
actor_net.zero_grad()
critic_net.zero_grad()
critic_loss.backward()
critic_opt.step()
```

$$Q_{target} = r_t + \gamma\hat{Q}(s_{t+1}, \hat{u}(s_{t+1}|\theta_{\hat{u}})|\theta_{\hat{Q}})$$

$$L = \frac{1}{N}\sum(Q_{target} - Q(s_t, a_t|\theta_Q))^2$$

target_q = r + gamma * next_q_values => 對應到第一條式子
criterion = nn.MSELoss() => 對應到第二條式子

- Explain effects of the discount factor. (5%)

Discount factor 大: 認為未來的事情影響目前的值多
Discount factor 小: 認為未來的事情影響目前的值少

- Explain benefits of epsilon-greedy in comparison to greedy action selection. (5%)

如果每次都挑最好的來選(Greedy)，可能會喪失發現更好的選擇的機會。
epsilon-greedy會讓agent有epsilon的機率去隨機選擇可能的下一個action

- Explain the necessity of the target network. (5%)

如果DQN訓練的時候更新的太過於頻繁，會讓訓練變得很不穩定。因為Q值還沒收斂，卻又拿來當成新的target value。因此要間隔一段steps再更新eval net到target net.

- Explain the effect of replay buffer size in case of too large or too small. (5%)

設得太大，很久以前玩的很爛的經驗也會拿出來重新學習
設得太小，顯得太目光狹隘，說不定好的action存在於已經被洗掉的buffer中

Performance:

DQN: 基本上都能達到 500 滿分

```python
def test(env, render, show=False):
    #print('Start Testing')
    epsilon = args.test_epsilon
    seeds = (20190813 + i for i in range(100))
    total_score = 0.0
    for seed in seeds:
        total_reward = 0
        env.seed(seed)
        state = env.reset()
        score = 0.0
        while(True):
            state_tensor = torch.Tensor(state).to(args.device)
            a = select_action(epsilon,state_tensor)
            next_state, reward, done, _ = env.step(a)
            score += reward
            state = next_state
            if show:
                #print('score : {:.0f}, action : {}, next_state
                env.render()
            if done:
                print(score)
                total_score += score
                break
    return total_score/ 10
```

max	avg	test count
500	500	100

DDPG: 練了大概 3500 epoch

```python
def test(env, render, show=False):
    #print('Start Testing')
    epsilon = args.test_epsilon
    seeds = (20190813 + i for i in range(100))
    total_score = 0.0
    for seed in seeds:
        total_reward = 0
        env.seed(seed)
        state = env.reset()
        score = 0.0
        while(True):
            state_tensor = torch.Tensor(state).to(args.device)
            a = select_action(epsilon,state_tensor)
            next_state, reward, done, _ = env.step(a)
            score += reward
            state = next_state
            if show:
                #print('score : {:.0f}, action : {}, next_state
                env.render()
            if done:
                print(score)
                total_score += score
                break
    return total_score/ 10
```

avg	max	min	test count
-153.8	-2.256	-533.589	100