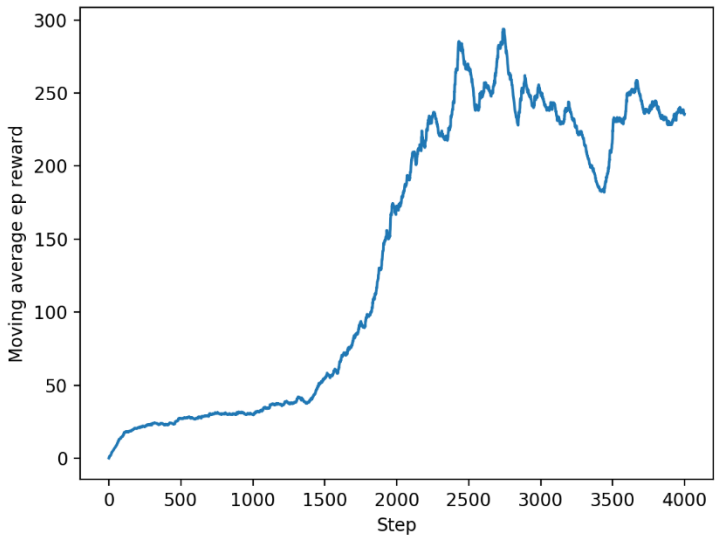
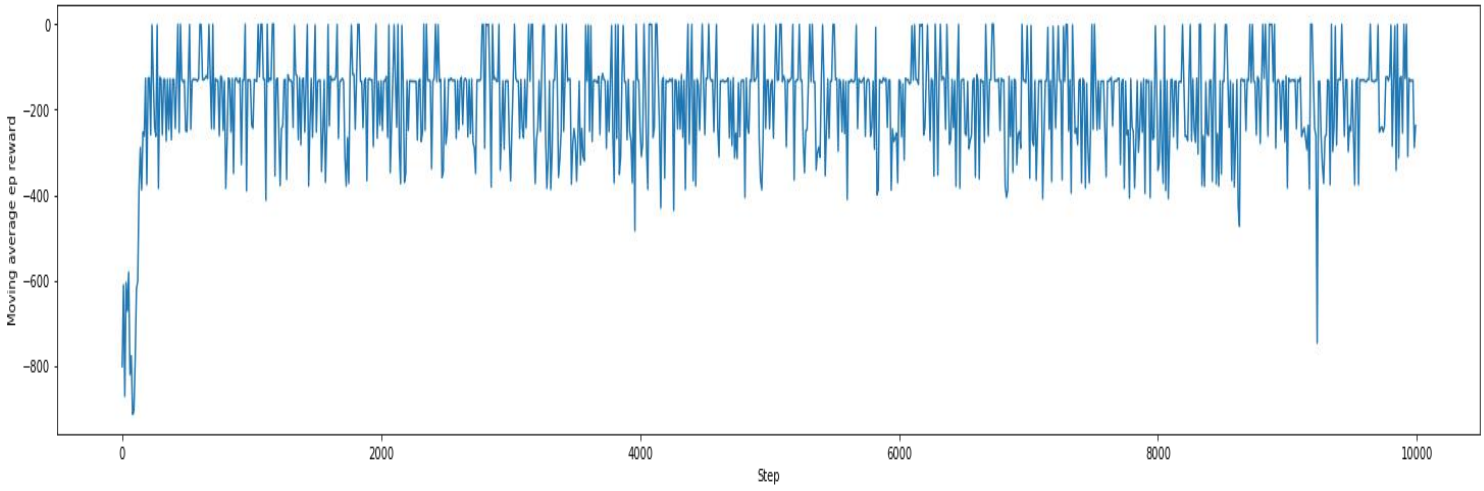
Lab 8: DQN and DDPG

- **A plot shows episode rewards of at least 1000 training episodes in the game CartPole-v0 (DQN)**



- **A plot shows episode rewards of at least 10000 training episodes in the game Pendulum (DDPG)**



- **Describe your implement/adjustment of the network structure & each loss function (DQN + DDPG)**

  1. **DQN**

     DQN可以利用neural network取代Q-learning用Q-table記憶state和action，好處是neural network有更好generalization的效果，可以從龐大的state space中自動提取特徵。其中我將Network都調整程簡單的Fully connected linear layer，參數用normal distribution的分布initilize到0~1之間，最後用relu當activation function輸出。

```python
class Net(nn.Module):
    def __init__(self, n_states, n_actions, n_hidden):
        super(Net, self).__init__()

        # 輸入層 (state) 到隱藏層，隱藏層到輸出層 (action)
        self.fc1 = nn.Linear(n_states, n_hidden)
        self.fc1.weight.data.normal_(0, 0.1)   # initialization
        self.out = nn.Linear(n_hidden, n_actions)
        self.out.weight.data.normal_(0, 0.1)   # initialization

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x) # ReLU activation
        actions_value = self.out(x)
        return actions_value
```

在Loss function方面，DQN是用mean square error計算evaluation network及target network的Q-value差距的平方，而這個想法也是由Q-Learning來的，Q-Learning的更新公式如下:

$$Q^*(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

而DQN的Loss function為:

$$L(\theta) = E[(TargetQ - Q(s,a;\theta))^2]$$

其中 θ 是神經網絡的參數，所以可以得知Target Q是:

$$TargetQ = r + \gamma \max_{a'} Q(s',a';\theta)$$

所以DQN是想要是當前的Q(s, a；θ)去逼近Target Q值，程式碼如下:

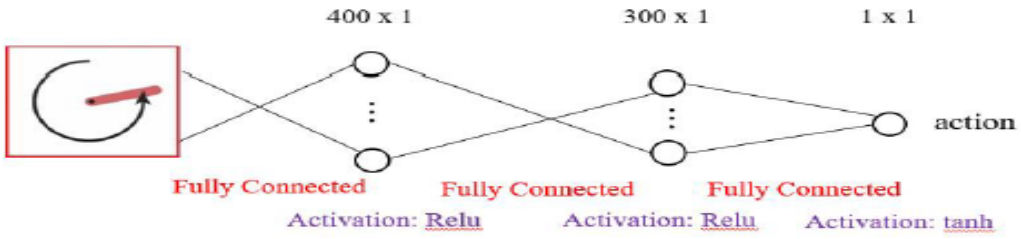
```python
# 計算現有 eval net 和 target net 得出 Q value 的落差
q_eval = self.eval_net(b_state).gather(1, b_action) # 重新計算這些 experience 當下 eval net 所得出的 Q value
q_next = self.target_net(b_next_state).detach() # detach 才不會訓練到 target net
q_target = b_reward + self.gamma * q_next.max(1)[0].view(self.batch_size, 1) # 計算這些 experience 當下 target net 所得出的 Q value
loss = self.loss_func(q_eval, q_target)

# Backpropagation
self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()
```

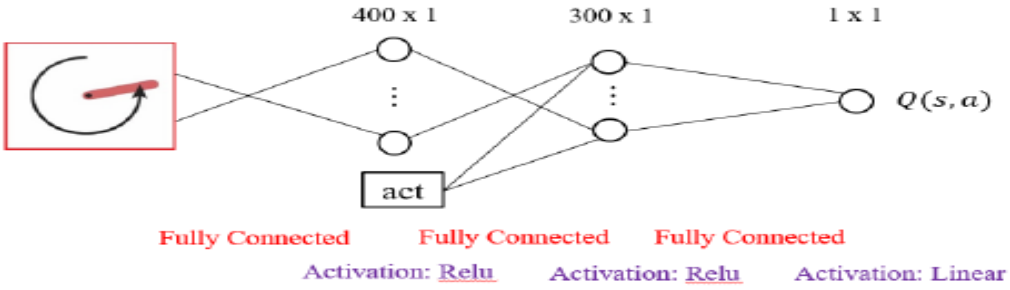
2. DDPG

DDPG 採用 actor-critic 架構，actor 輸出動作，critic 評判動作後輸出價值，並且借鑑 DQN 的思想，所以共有四個神經網絡。即: critic 部分有兩個神經網絡，target network Q' 和 critic network Q；actor 部分有兩個神經網絡：target network u' 和 actor network u。首先 actor 和 critic 的架構如講義。在 actor 的部分 actor network u 的輸入為當前狀態，target network u'的輸入為下一狀態；在 critic 的部分 critic network Q 的輸入為當前狀態和當下動作，target network Q'的輸入為下一狀態和下一動作。兩部分的 hidden layer 也都分別是 400 和 300 並以 fully connected layer，最後 actor 輸出 action，critic 輸出 Q-value 值。

- Actor



- Critic



- **Describe how you implement the training process of deep Q-learning (DQN)**

  在訓練DQN的過程中，要設定max_steps以比免遊戲無法終止，在每一輪一開始的時候，會先刷新環境後再選擇action，依epslion的機率決定是否要依照最大的Q-value值走，之後就執行行動返回下一個狀態和reward等資訊，之後修改reward使得agent可以更抓到遊戲的訣竅，分別是使車子的水平位移變小，並讓棒子越正越好，儲存經驗 (儲存Q-value值)，直到有足夠的experience後再進行訓練，接著進入到下一個state，反覆循環直到遊戲終止或是達到max_steps。

```
# 學習
for step, i_episode in enumerate(range(n_episodes)):
    t = 0
    rewards = 0
    state = env.reset()
    for episode in range(max_steps):
        env.render()

        # 選擇 action
        action = dqn.choose_action(state)
        next_state, reward, done, info = env.step(action)

        # 修改 reward，加快訓練
        x, v, theta, omega = next_state
        r1 = (env.x_threshold - abs(x)) / env.x_threshold - 0.8 # 小車離中間越近越好
        r2 = (env.theta_threshold_radians - abs(theta)) / env.theta_threshold_radians - 0.5 # 柱子越正越好
        reward = r1 + r2

        # 儲存 experience
        dqn.store_transition(state, action, reward, next_state)

        # 累積 reward
        rewards += reward

        # 有足夠 experience 後進行訓練
        if dqn.memory_counter > memory_capacity:
            dqn.learn()

        # 進入下一 state
        state = next_state

        if done:
            collection_rewards.append(reward)
            print('{}/1000 episode finished after {} timesteps, total rewards {}'.format(step+1, t+1, reward))
            break

        t += 1
```

- **Describe the way you implement of epsilon-greedy action select method (DQN)**
  在這一次程式碼裡面的epsilon = 0.1，若是用np.random.uniform的方式去生成的值小於0.1，那agent會隨機往左或往右移 (carpole)，如此便有一定的機率會去走沒有走過的路；反之，若是用np.random.uniform生成的值大於1，agent便會根據Q-value最大的值去做出action。

```
def choose_action(self, state):
    x = torch.unsqueeze(torch.FloatTensor(state), 0)

    # epsilon-greedy
    if np.random.uniform() < self.epsilon: # 隨機
        action = np.random.randint(0, self.n_actions)
    else: # 根據現有 policy 做最好的選擇
        actions_value = self.eval_net(x) # 以現有 eval net 得出各個 action 的分數
        action = torch.max(actions_value, 1)[1].data.numpy()[0] # 挑選最高分的 action

    return action
```

- **Explain the mechanism of critic updating (DDPG)**
  在 DDPG critic 部分對參數 θ_Q 的更新，是採用 DQN 中 TD error 的 Loss function:

$$L(\theta^Q) = \mathbb{E}_{\mu'}\left[\left(Q(s_t, a_t|\theta^Q) - y_t\right)^2\right]$$

  其中，

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$$

- **Explain the mechanism of actor updating (DDPG)**
  actor參數的更新涉及到critic，其中grad[Q]代表受到critic影響，而後半自身的 θ_u代表actor會將自身參數optimization，整合在一起就是actor會朝著Maxium Q-value的方向更新參數。

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

- **Describe how to calculate the gradients? (DDPG)**
  對於計算actor的policy gradient是利用David Sliver在2014年提出DPG（Deterministic Policy Gradient，DDPG改善DQN策略動作以stochastic去執行動作，變成是以derministic的方式去執行動作，使得Q-value的Bellman function由:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}\sim E, a_t\sim\pi}\left[r(s_t, a_t) + \gamma\,\mathbb{E}_\pi\left[Q^\pi(s_{t+1}, a_{t+1})\right]\right]$$

  改為:

$$Q^\mu(s_t, a_t) = \mathbb{E}_{s_{t+1}\sim E}\left[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))\right]$$

- **Describe how the code work (the whole code) (DQN + DDPG)**

  1. DQN

  DQN一開始一樣要建立environment，先隨機選擇action後修改reward以加快訓練，將state、action、下一個state以及修改後的reward存進memory以便進行之後的replay memory，一直進型訓練直到agent訓練到一定量的epoch後才開始讓dqn model更新 (利用q_eval和q_target的MSE更新q_eval_net)，之後在依固定間隔更新target_net的參數 (藉由複製q_eval_net的參數到

target_net)，下一階段則依照epsilon-greedy 選擇action進行循環直到遊戲結束或epoch循環完畢。

2. DDPG

DDPG一開始一樣要建立environment，先隨機選擇action並為了之後的exploration在action上增加randomness，得到下一步狀態的資訊後先儲存在replay memory buffer中作為online training的dataset，之後會隨機sampling N個 transition data作為一個mini-batch的training data，此時可以得知更新actor和critic網絡參數的狀態、動作、reward、下一狀態。再來分別更新actor network和critic network的網絡參數，最後依據soft update以更新target network的參數 (Q-target和現有的critic network output Q-value)。

- **Other study or improvement for the project.**

  在DQN的訓練過程中，要訓練柱子要立起來不能倒下，最intial的設定是柱子正立的時間越久reward越高，但是在訓練過程中卻發現agent會亂移動導致遊戲常常失敗，為了增加model的intuition，所以在reward的地方，改為設定成小車離中間越近的時候，和柱子越正時reward越高，以增加訓練速度和效率。

- **Performance**

  1. DQN: 由於訓練訓練較多場 (4000場)，後期reward普遍較高，平均每一百輪的testing episodes為213.5，除以2後是106.75。
  2. DDPG: 大約到 epoch = 300 後，每輪的 reward 大約接落在 0~400 之間，其中單輪最高的 reward 是-0.0294，加 700 再除以 5 後是 139.99412