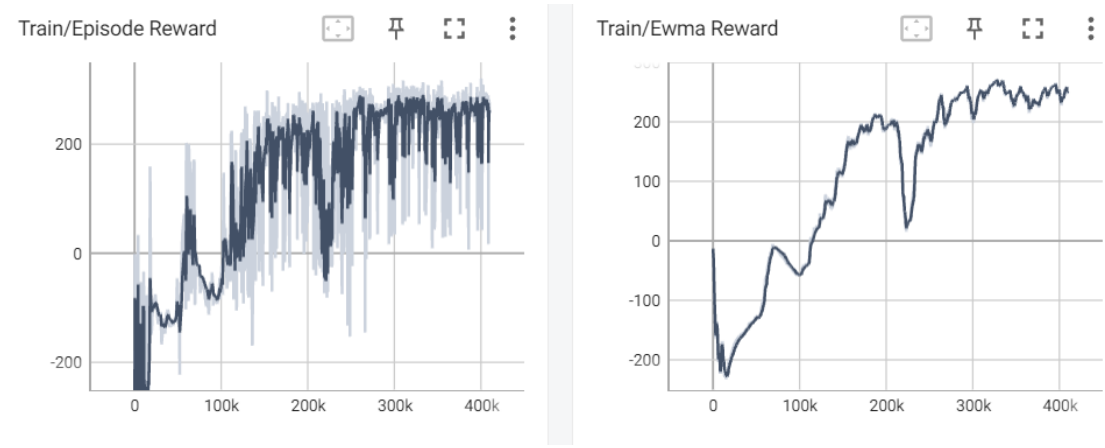
Lab5

1. Experimental Results

(1) Dqn

```
PS C:\Users\user\Desktop> python dqn.py --test_only
C:\Users\user\AppData\Local\Programs\Python\Python38\lib\site-packages\gym\logger.py:30: UserWarning: WARN: Box bound precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode 1: 251.36439770479876
Episode 2: 282.25070074327357
Episode 3: 282.3655744170303
Episode 4: 278.61136905740113
Episode 5: 192.65664535868748
Episode 6: 265.1129661029304
Episode 7: 304.38402345049667
Episode 8: 299.058513397492
Episode 9: 301.61448481194213
Episode 10: 294.48963194193004
Average Reward 275.19083069859823
```

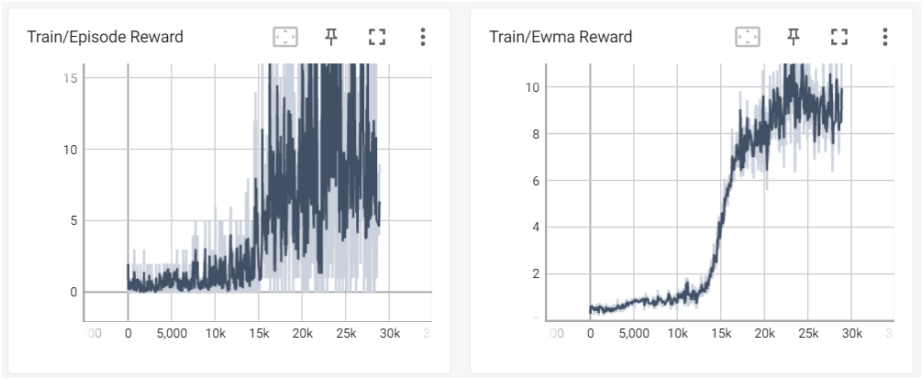


(2) Ddpg

```
PS C:\Users\user\Desktop> python ddpg.py --test_only
C:\Users\user\AppData\Local\Programs\Python\Python38\lib\site-packages\gym\logger.py:30: UserWarning: WARN: Box bound precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
episode 1: 245.24
episode 2: 262.81
episode 3: 272.55
episode 4: 271.59
episode 5: 250.48
episode 6: 200.40
episode 7: 301.73
episode 8: 269.91
episode 9: 303.07
episode 10: 244.41
Average Reward 262.2188486836336
```

(3) dqn_breakout

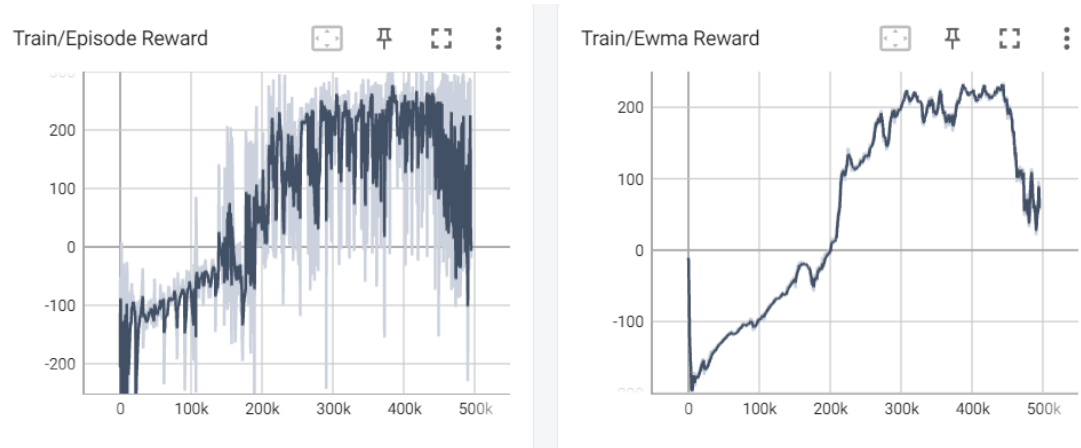
```
PS C:\Users\user\Desktop> python dqn_breakout.py --test_only
Start Testing
episode 1: 259.00
episode 2: 210.00
episode 3: 176.00
episode 4: 324.00
episode 5: 299.00
episode 6: 307.00
episode 7: 383.00
episode 8: 207.00
episode 9: 311.00
episode 10: 317.00
Average Reward: 279.30
```



2. Experimental Results of bonus parts

(1) Ddqn

```
PS C:\Users\user\Desktop> python ddqn.py --test_only
C:\Users\user\AppData\Local\Programs\Python\Python38\lib\site-packages\gym\logger.py:30: UserWarning: WARN: Box bound precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
Episode 1: 177.5826454393235
Episode 2: 221.0350648834515
Episode 3: 268.21493011148755
Episode 4: 276.6417783332877
Episode 5: 255.71792398341543
Episode 6: 137.7029322110045
Episode 7: 144.73057056432597
Episode 8: 284.85811759721486
Episode 9: 295.49846723499115
Episode 10: 304.05880772873087
Average Reward 236.60412380872327
```



(2) td3

```
PS C:\Users\user\Desktop> python td3.py --test_only
C:\Users\user\AppData\Local\Programs\Python\Python38\lib\site-packages\gym\logger.py:30: UserWarning: WARN: Box bound precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))
Start Testing
episode 1: 247.75
episode 2: 267.16
episode 3: 284.60
episode 4: 269.61
episode 5: 267.76
episode 6: 259.87
episode 7: 261.41
episode 8: 233.77
episode 9: 280.19
episode 10: 274.30
Average Reward 264.6411252631321
```

3. Questions

- Describe your major implementation of both DQN and DDPG in detail. Your description should at least contain three parts:
  (1) Your implementation of Q network updating in DQN

```python
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=64):
        super().__init__()
        self.block1 = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
        )

        self.block2 = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
        )

        self.block3 = nn.Sequential(
            nn.Linear(hidden_dim, action_dim),
        )

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        return x
```

這個部分我設計了三個線性層，並且 activation function 使用 relu，input 是 state，output 是 action，hidden layer 的維度設 64。

(2) Your implementation and the gradient of actor updating in DDPG

```
action = self._actor_net(state)
actor_loss = -self._critic_net(state, action).mean()
# optimize actor
actor_net.zero_grad()
critic_net.zero_grad()
actor_loss.backward()
actor_opt.step()
```

首先我們使用 actor net 來估計在給定狀態下應該採取的行動，接著我們用 critic net 評估在這給定狀態下採取該行動的價值或 Q 值，然後我們希望最大化這個價值，但因為優化過程通常是最小化 loss，所以我們使用負號來反轉問題。之後我們清除之前計算的梯度，因為在 PyTorch 中梯度是會累積的。然後根據計算的 actor loss 執行反向傳播來計算參數的梯度。最後使用定義的 actor 優化器來更新 actor net 的參數，這步驟將利用之前計算的梯度來進行參數更新，使 actor net 在未來能夠做出更好的行動選擇。

(3) Your implementation and the gradient of critic updating in DDPG

```
q_value = self._critic_net(state, action)
with torch.no_grad():
    a_next = self._target_actor_net(next_state)
    q_next = self._target_critic_net(next_state, a_next)
    q_target = reward + gamma * q_next * (1.0 - done)

criterion = nn.SmoothL1Loss()
critic_loss = criterion(q_value, q_target)
# optimize critic
actor_net.zero_grad()
critic_net.zero_grad()
critic_loss.backward()
critic_opt.step()
```

首先用_critic_net 評估在當前的狀態和行動下的價值估計，得到 q_value。接著使用_target_actor_net 預測下一個狀態的行動 a_next，並進一步使用_target_critic_net 來評估這個行動在下一個狀態下的 q_next。然後使用這個 q_next 以及 reward 和 gamma 計算 q_target。如果該階段已經結束（由 done 表示），則不將折扣後的未來價值加入到 q_target 中。接下來使用 nn.SmoothL1Loss()作為損失函數來計算 q_value 與 q_target 之間的差異，得到 critic_loss。最後，我們先重置 actor 和 critic net 的梯度，然後進行反向傳播以計算梯度，並使用 critic_opt 優化器更新 critic net 的參數。這段程式碼的目的是根據當前的狀態和行動，以及接下來的狀態和預測的行動，更新 critic net，使其能夠更準確地評估行動的價值。

■ Explain effects of the discount factor

當 discount factor 接近 1 時，代表學習模型會非常重視未來的 reward，這可能意味著模型在短期內會願意犧牲一些即時 reward，以期待在未來得到更大的回報。相反地，當 discount factor 接近 0 時，模型主要只會

關注當前或是很近期的 reward，對於遠期的 reward 則不太在意。所以，透過調整 discount factor 的大小，我們可以影響模型在追求即時 reward 和未來 reward 之間的平衡。此外，選擇不同的 discount factor 還可能會影響模型學習的穩定性和速度。太高的 discount factor 可能會導致學習在某些情境下變得不太穩定。

- Explain benefits of epsilon-greedy in comparison to greedy action selection

  epsilon-greedy 策略在選擇動作時結合了探索（exploration）和利用（exploitation）。這意味著 agent 有 ϵ 的機率去選擇一個隨機的動作（即探索），而有 $1-\epsilon$ 的機率去選擇目前認為最佳的動作（即利用）。這種策略的好處是它確保了 agent 不僅僅基於目前的知識進行動作選擇，還會嘗試新的動作來更好地了解環境。相比之下，純粹的 greedy action selection 總是選擇當前認為最佳的動作，也就是 q value 最大的動作。這種策略完全基於利用，沒有探索的部分。雖然這在某些情況下可能是有效的，但如果 agent 的初始知識不完整或有誤，它可能會錯過更好的策略或獎勵。

- Explain the necessity of the target network

  在 DQN 中，使用同一個 network 計算 current 和 future Q-values 可能會使 training 不穩定，因為更新 weights 會同時影響兩者。為了解決這個問題，我們引入了 target network。這個 network 的 weights 較少更新，從而提供更穩定的 future Q-value 估計。使用 target network 可以使 training 過程更加穩定，因為它減少了目標 Q-values 的快速變動。

- Describe the tricks you used in Breakout and their effects, and how they differ from those used in LunarLander

  在 Breakout 中，我設 Episode Life = True，讓 agent 每次失去生命後結束遊戲是有意義的，因為這可以讓 agent 學到失去生命的重要性。在 LunarLander 中，這樣的設置可能不是很關鍵，因為它本身已經有機制來懲罰墜毀。至於輸入方面，Breakout 的是圖像，所以需要將最後四個 frames 進行堆疊來捕捉動態，所以我設 Stack Frame = True，並調整輸入的 shape 以適應 CNN 模型。LunarLander 的輸入則是各種物理狀態，不需要進行 frame 堆疊。