HW3 Report

姓名：黃浩恩
學號：f05921120
Coworker：賴棹沅、謝忱、嚴聲揚

Q1 Basic Performance (6%)

Q1.1 Describe your Policy Gradient & DQN model (1% + 1%)

Q1.1.a Policy Gradient

Policy Gradient 的 model，是透過 NN 將 state 當作輸入，action 當作輸出，使 NN 直接訓練目前的 state 對應的 action，不分析 Q 值，透過 Categorical 的 distributions 直接輸出行為，並學習調透過 reward 的反應調高或降低目前選擇該 action 的機率。

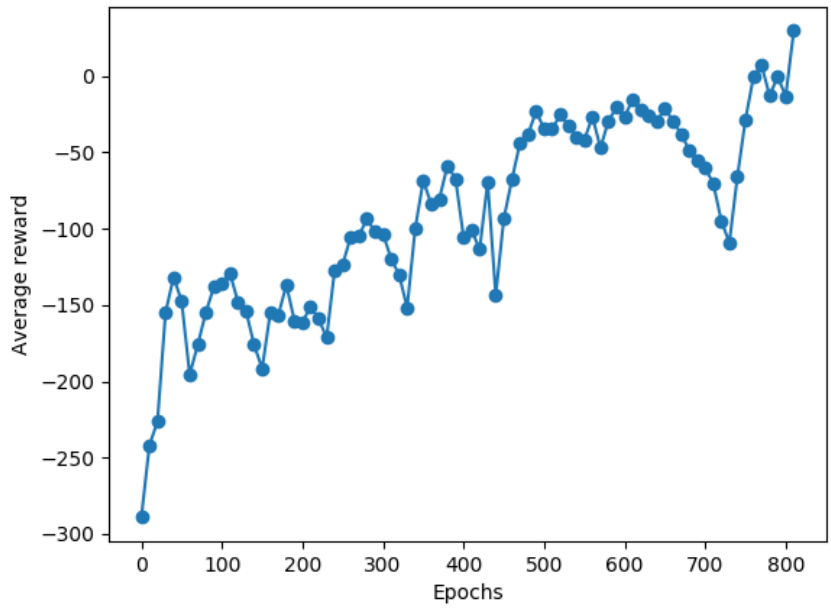
```
class PolicyNet(nn.Module):
    def __init__(self, state_dim, action_num, hidden_dim):
        super(PolicyNet, self).__init__()
        self.fc1 = nn.Linear(state_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, action_num)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        action_prob = F.softmax(x, dim=1)
        return action_prob
```

Q1.1.b DQN model

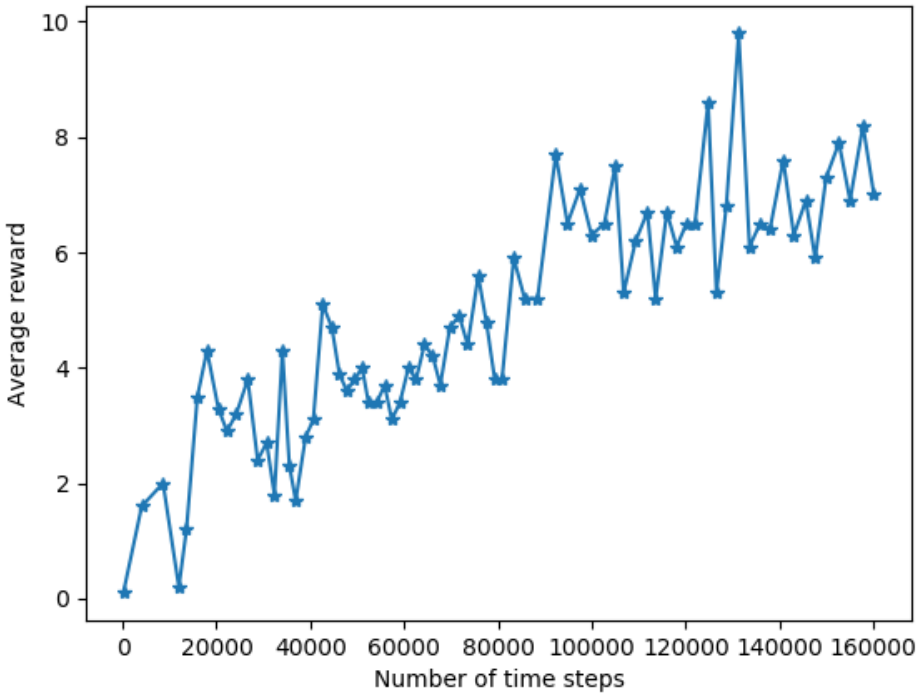
DQN 一開始會先有個隨機取樣機率 Epsilon，透過該機率可以選擇隨機選擇任意動作或是進入 NN。隨機選擇動作是因為讓 NN 透過 ReplayMemory 歷史資料學到不同 state 下各動作的 Q 值。倘若進入 NN，DQN model 的 NN 架構事先透過 CNN 將 state 資料採樣，後再將輸出與 action 連結。得到每個 action 的 Q 值後，再透過 max 取出最大 Q 值的動作，整個運作過程就是 Value base 的運作方式。其中因為 Q-learning 是一種 off-policy 的離線學習，可以學習 online 與過往歷史經歷，透過 ReplayMemory 將過去玩過的歷史資料存取下來，隨機抽取來學習打斷資料連續相關性的問題(target_net)。

```python
class DQN(nn.Module):
    """
    This architecture is the one from OpenAI Baseline, with small modification.
    """
    def __init__(self, channels, num_actions):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(channels, 32, kernel_size=8, stride=4)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1)
        self.fc = nn.Linear(3136, 512)
        self.head = nn.Linear(512, num_actions)

        # modify 2
        # self.conv1 = nn.Conv2d(channels, 32, kernel_size=8, stride=4)
        # self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)
        # self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1)
        # self.fc1 = nn.Linear(3136, 1024)
        # self.fc2 = nn.Linear(1024, 512)
        # self.head = nn.Linear(512, num_actions)

        self.relu = nn.ReLU()
        self.lrelu = nn.LeakyReLU(0.01)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.lrelu(self.fc(x.view(x.size(0), -1)))
        q = self.head(x)
        return q
```

Q1.2 Plot the learning curve to show the performance of your Policy Gradient on LunarLander (2%)



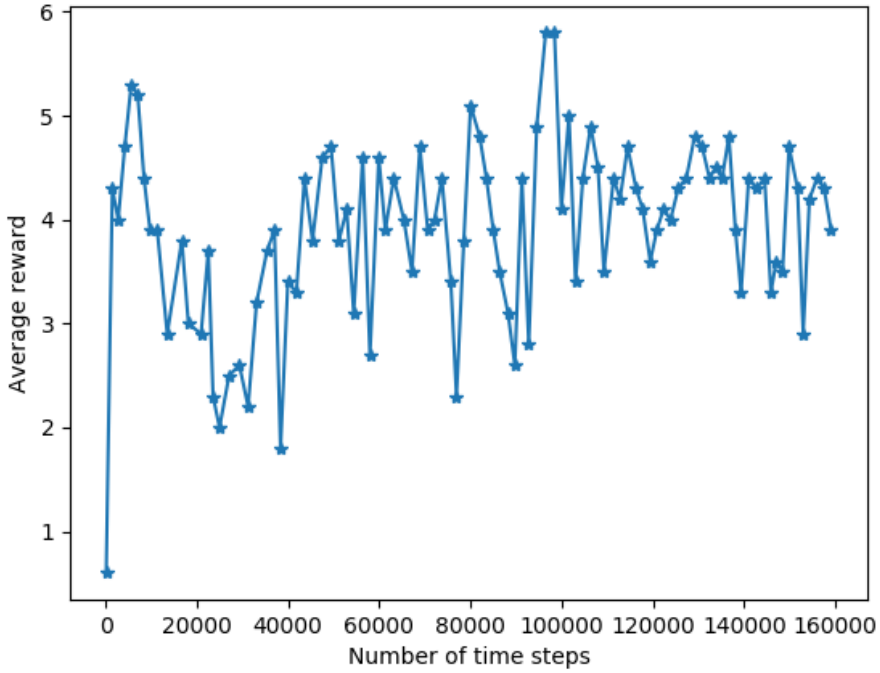
Q1.3 Plot the learning curve to show the performance of your DQN on Assualt (2%)


Average reward
Number of time steps
0
2
4
6
8
10
0
20000
40000
60000
80000
100000
120000
140000
160000

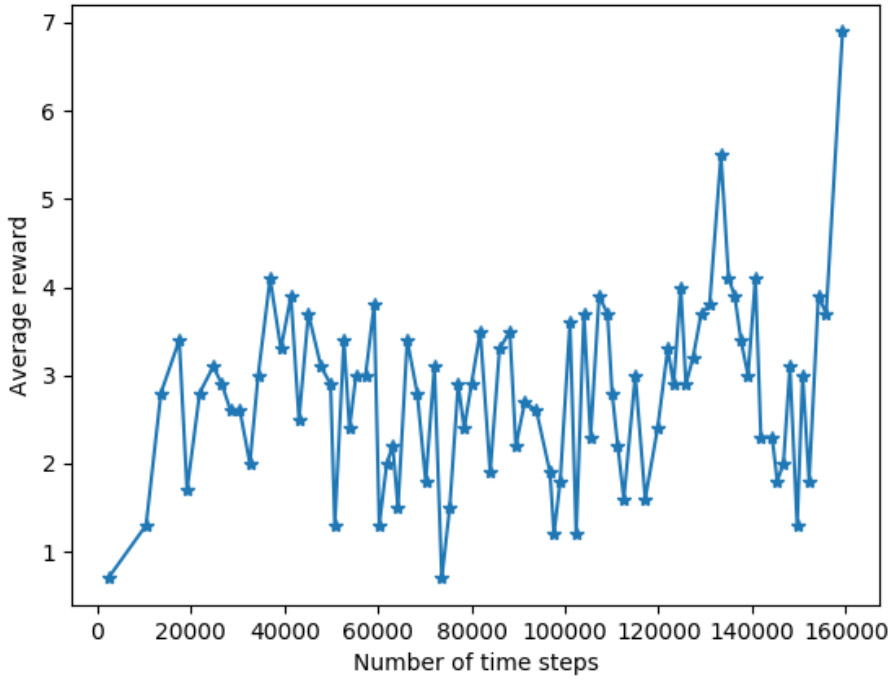
Q2 Experimenting with DQN hyperparameters (2%)

Q2.1 Plot all four learning curves in the same figure (1%)

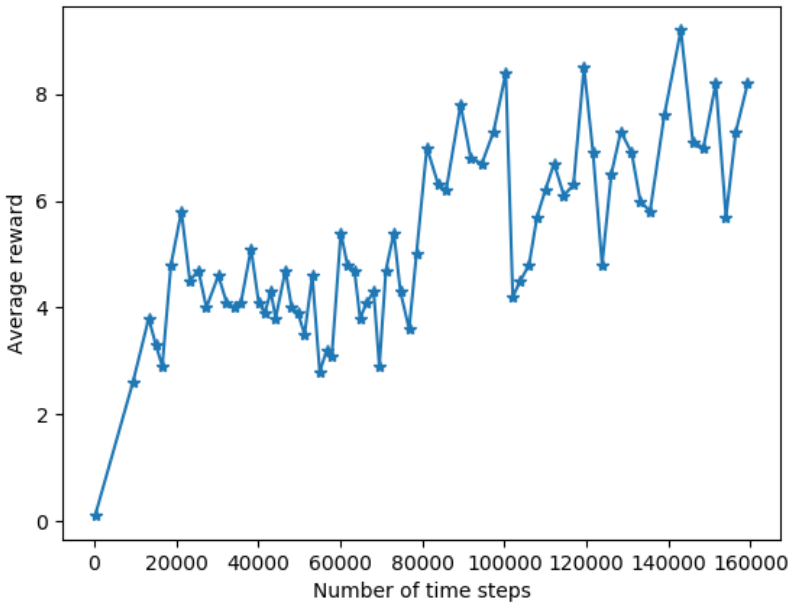
Q2.1.a 更改 GAMMA



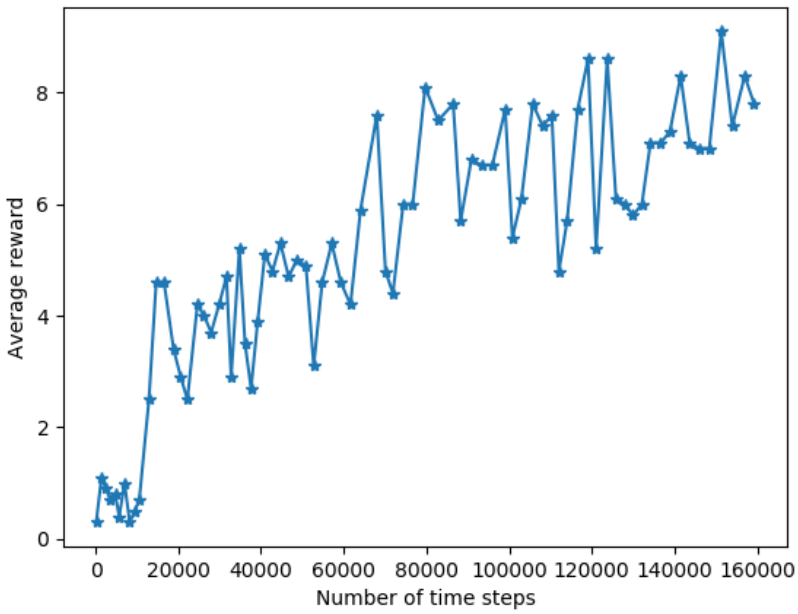
Q2.1.b 將 replay memory 從 10000 調低為 100



Q2.1.c 將 DQN 神經架構做更改，增加一層 Linear



Q2.1.d　　提升 target_update_freq 與 batch_size 使分數提升較快



Q2.2 Explain why you choose this hyperparameter and how it affect the results (0.5% + 0.5%)

Q2.2.a 更改 GAMMA，使得原本的 GAMMA=0.99 變為 GAMMA=0.75，讓學習歷史資料的衰減率增加。由圖可見，學習的平均獎勵以 0.99 優於 0.75，表示歷史學習之關係度仍為重要。

Q2.2.b 將 replay memory 從 10000 調低為 100，由圖可見，效果非常不好，原因可能為存在 memory 的資料，即使隨機選取的資料關聯度依然太高，無法從更多不同之 state 與 action 中學到資訊。

Q2.2.c 將 DQN 神經架構做更改，增加一層 Linear 於 CNN 接至 Linear 後，來觀察原先助教提供的架構之神經元個數的差距跳太大與較緩和的下降的比較。由圖可見，平均分比起原先得來的高，可能是透過該層 Linear 擷取了更多的資訊，在將 state mapping 到 action 上做得更加完整。

Q2.2.d 提升 target_update_freq，在參數調整上從原本的 1000 調降至 200，使 200 步就更新一次，並再將 batch_size 提升一倍，從 32 調至 64，使訓練速度上升速度增快，再訓練過程中，以上 3 種更改方法皆耗時約 20 分鐘，但此次僅 18 分鐘就完成。由圖可知，進步效率較快，於最後的平均分數也與原本的差不多，效果不錯。

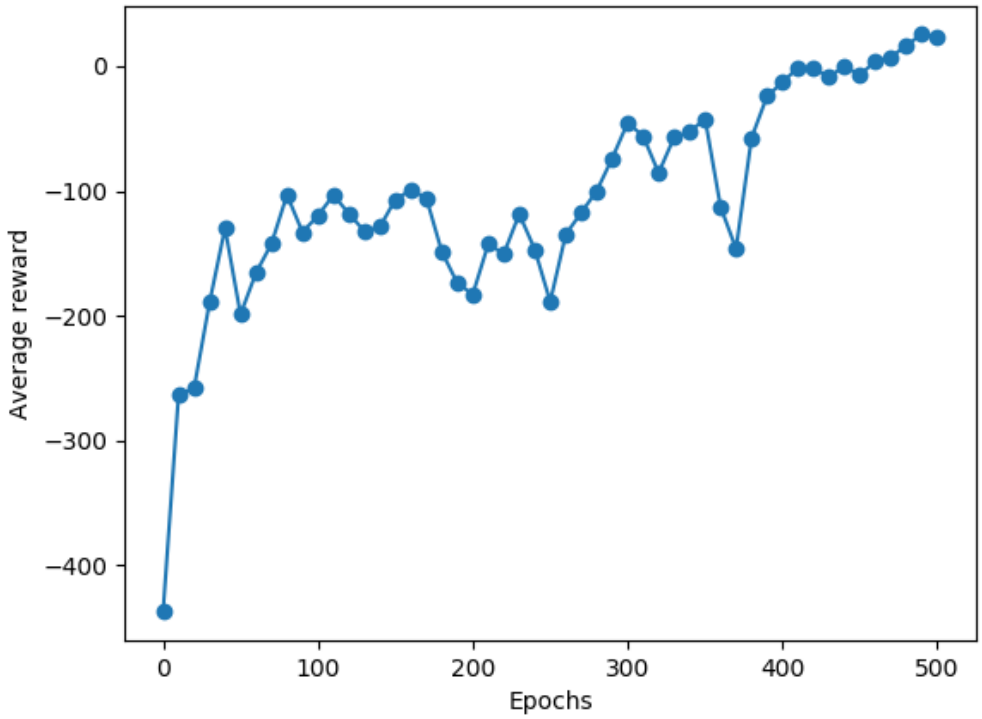
Q3 Improvements to Policy Gradient & DQN / Other RL methods (2% + 2%)

Q3.1 describe why they can improve the performance (1%)

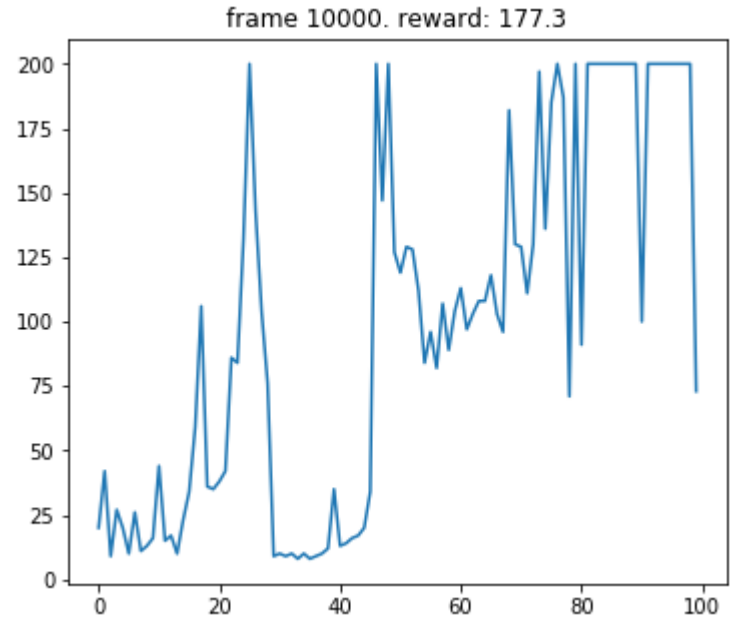
Actor-Critic 是一個結合 value-base 與 policy-base 的架構。針對 policy-base 部分，以每回合結束來更新一次，但是 value-base 的學習卻可以很快速的更新，除了可以針對連續動作來做訓練、學習外，亦可以像一個眼睛，看著 action 的動作、reward 的好壞，來進行超脫訓練的學習。

Q3.2 Plot the graph to compare results with and without improvement (1%)

Q3.2.a **將 LunarLander-v2 用 A2C 來做 improvements，可以看到如果照著作業的目標，當 avg_reward > 50 就停止的話，則原先的 PG 需要花上約 800 個 step 才能達到目標，但是使用 A2C 的話，僅需要 500 個 step 就可以達到目標。可見訓練速度幾乎快上 PG 要上 1.6 倍。**



Q3.2.b 傳統的 DQN 普遍會過高估計 Action 的 Q 值，而且估計誤差會隨 Action 的個數增加而增加。如果高估不是均勻的，則會導致某個次優的 Action 高估的 Q 值超過了最優 Action 的 Q 值，永遠無法找到最優的策略。在許多基於視覺的感知的 DRL 任務中，不同的狀態動作對的值函數是不同的，但是在某些狀態下，值函數的大小與動作無關。於”CartPole-v0”遊戲中的結果如下。



他的 improvement 之 DDQN 結果如下：