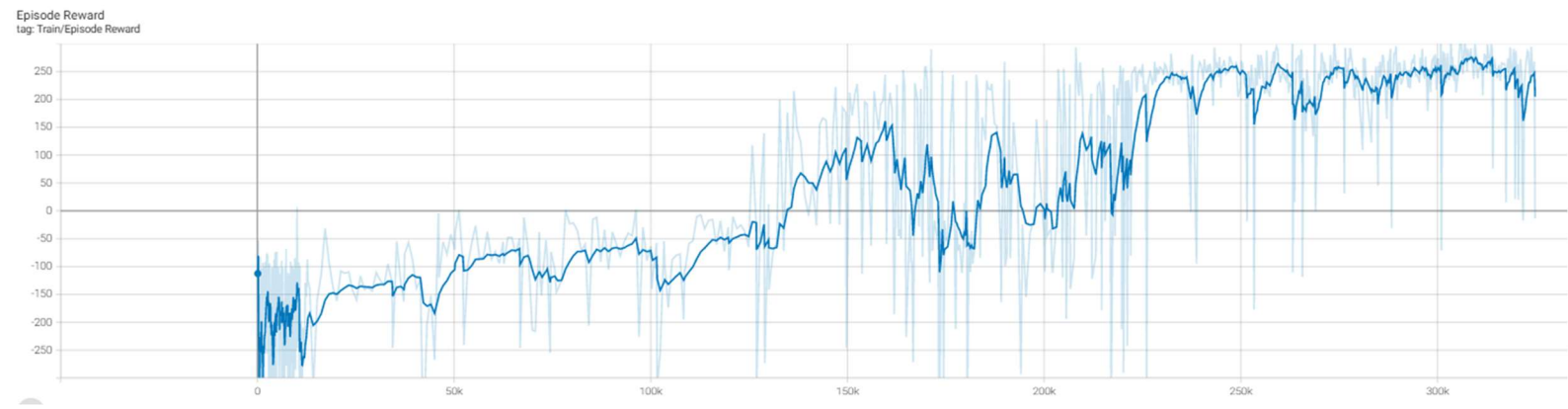
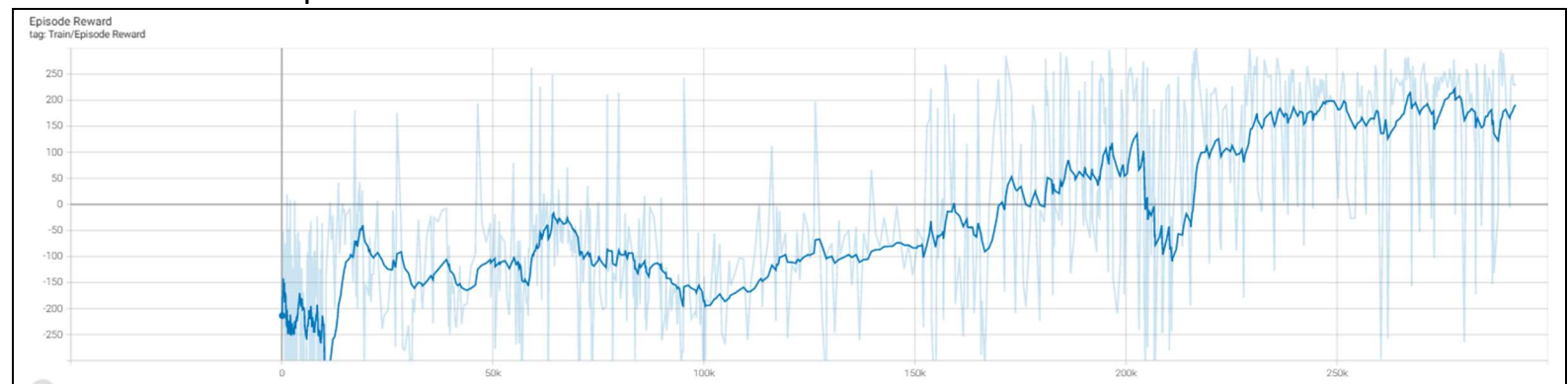


Report Spec

1. A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLander-v2



2. A tensorboard plot shows episode rewards of at least 800 training episodes in LunarLanderContinuous-v2

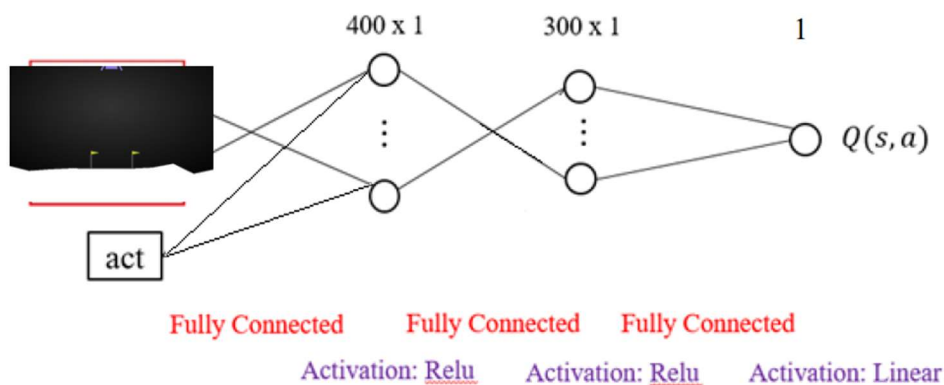


3. Describe your major implementation of both algorithms in detail.

DQN :

建立 network 來預測 $Q(S,a)$ 的值 (state 加上 action 後的值) 其中 Input 為經過 action 後的 state, output 為 4 個 action (no-op, fire left engine, fire main engine, fire right engine), network 如下:

- Critic



```
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=32):
        super().__init__()
        ## TODO ##
        self.lin1 = nn.Sequential(
            nn.Linear(state_dim, 400),
            nn.ReLU()
        )
        self.lin2 = nn.Sequential(
            nn.Linear(400, 300),
            nn.ReLU()
        )
        self.lin3 = nn.Sequential(
            nn.Linear(300, action_dim),
            # nn.ReLU()
        )
```

資料收集的部分：

- 選擇 action

With probability ϵ select a random action a_t
 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

```
def select_action(self, state, epsilon, action_space):
    '''epsilon-greedy based on behavior network'''
    ## TODO ##
    if random.random() < epsilon:
        return action_space.sample()
    else:
        with torch.no_grad():
            return self._behavior_net(torch.tensor(state).to(self.device)).topk(1)[1].item()
```

這裡為了怕 overfit 所以我們是從 behavior_net 中取出最適合的 action。

- 利用 append 來收集之後訓練所需資料

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

```
def append(self, state, action, reward, next_state, done):
    self._memory.append(state, [action], [reward / 10], next_state,
                        [int(done)])
```

訓練的部分：

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

```
q_value = self._behavior_net(state).gather(dim=1, index=action.to(dtype = int))
with torch.no_grad():
    q_next = self._target_net(torch.tensor(next_state).to(self.device)).topk(1)[0]
    q_target = q_next*gamma*(1-done) + reward
```

從前面 append 的資料中取一個 state 與下個 state 之間的變化，

做為訓練用的 data (TD-learning 精隨)

其中利用本次 state 的 value(來自 behavior_net) 下個 state 的 value(來自 target_net)

```
criterion = nn.MSELoss()
loss = criterion(q_value, q_target)
```

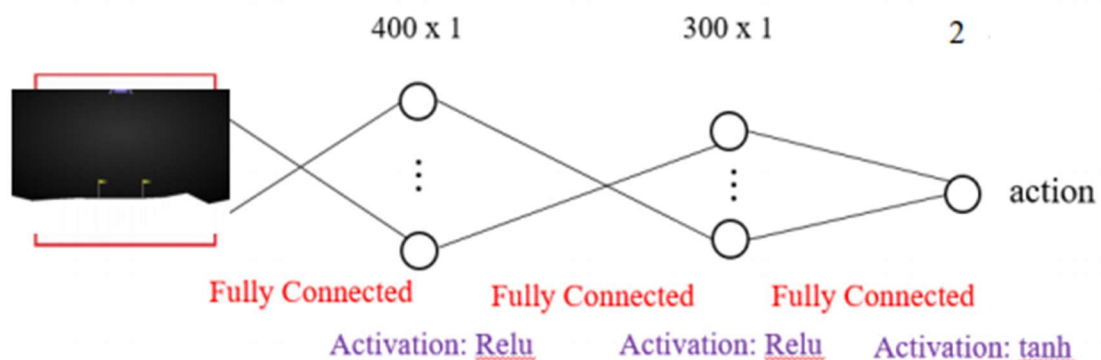
再利用 MSE 取得與 loss

```
def _update_target_network(self):
    '''update target network by copying from behavior network'''
    ## TODO ##
    self._target_net.load_state_dict(self._behavior_net.state_dict())
    # raise NotImplementedError
```

最後每隔一段時間更新 target_network

DDPG:

- Actor



```

class ActorNet(nn.Module):
    def __init__(self, state_dim=8, action_dim=2, hidden_dim=(400, 300)):
        super().__init__()
        ## TODO ##
        self.lin1 = nn.Sequential(
            nn.Linear(state_dim, hidden_dim[0]),
            nn.ReLU()
        )
        self.lin2 = nn.Sequential(
            nn.Linear(hidden_dim[0], hidden_dim[1]),
            nn.ReLU()
        )
        self.lin3 = nn.Sequential(
            nn.Linear(hidden_dim[1], action_dim),
            nn.Tanh()
        )

    def forward(self, x):
        ## TODO ##
        out = self.lin1(x)
        out = self.lin2(out)
        out = self.lin3(out)
        return out

```

建立一個可以依據目前 state 決定要執行哪個 action 的 Actor Network, 其中 input 為目前的遊戲畫面, 另外 output 為 2 個 action (Main engine:-1~+1, Left-right-engine:-1~+1)

```

class CriticNet(nn.Module):
    def __init__(self, state_dim=8, action_dim=2, hidden_dim=(400, 300)):
        super().__init__()
        h1, h2 = hidden_dim
        self.critic_head = nn.Sequential(
            nn.Linear(state_dim + action_dim, h1),
            nn.ReLU(),
        )
        self.critic = nn.Sequential(
            nn.Linear(h1, h2),
            nn.ReLU(),
            nn.Linear(h2, 1),
        )

    def forward(self, x, action):
        x = self.critic_head(torch.cat([x, action], dim=1))
        return self.critic(x)

```

另外我們還是需要一個計算目前 action 後 state 的值 所以需要此 network 輸出 1 output

Select action $a_t = \mu(s_t | \theta^\mu) + N_t$ according to the current policy and exploration noise

```

def select_action(self, state, noise=True):
    '''based on the behavior (actor) network and exploration noise'''
    ## TODO ##
    with torch.no_grad():
        if noise:
            re = self._actor_net(torch.tensor(state).to(self.device)) + \
                torch.tensor(self._action_noise.sample()).to(self.device)
        else:
            re = self._actor_net(torch.tensor(state).to(self.device))
    return re.cpu().numpy().squeeze()

```

另外在玩遊戲過程中，需在 action 上加上 noise

```
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-done)
criterion = nn.MSELoss()
critic_loss = criterion(q_value, q_target)
```

利用 Target net 生出的 q_{target} 與 Behavior net 生出的 q_{value} 做 MSE Loss 以用來更新 Critic net 判斷的值，使其越來越正確。

```
## update actor ##
# actor loss
## TODO ##
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()
```

最後利用 Behavior net 的 Actor net μ 與 Critic Network Q 可以求出 $Q(s,a)$ ，另外我們需要 $Q(s,a)$ 越大越好，因此定義 Loss Value = $E[-Q(s, \mu(s))]$ 進行更新

4. Describe differences between your implementation and algorithms.

最主要的差別為，在實作中有個 warmup 的部分，在這段時間中主要用途為收集資料，並不會去更新 behavior or target network 中的值。

algo 中寫法則是寫說每次個 episode 還會重新收集一次資料，並且存入 memory 中。

5. Describe your implementation and the gradient of actor updating.

利用 Behavior Actor net μ 與 Critic net Q 可以求出 $Q(s,a)$ ，我們想更新 Actor net μ 來使輸出的 $Q(s,a)$ 越大越好，因此定義 Loss Value = $-Q(s, \mu(s))$ ，另外 backpropagation 的時候不更新 Critic、只更新 Actor


```

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()

```

6. Describe your implementation and the gradient of critic updating.

利用 Target Network 生出的 Q_{target} 與 Behavior Network 生出的 $Q(s,a)$ 做 mean square error 來更新 Q Network。

```

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-done)
criterion = nn.MSELoss()
critic_loss = criterion(q_value, q_target)

# optimize critic
actor_net.zero_grad()
critic_net.zero_grad()
critic_loss.backward()
critic_opt.step()

```

7. Explain effects of the discount factor.

$$G_t = R_{t+1} + \lambda R_{t+2} + \dots = \sum_{k=0}^{\infty} \lambda^k R_{t+k+1}$$

表示越前面的 reward 對 Ground Truth 所造成的影響是越大的，而反之離 Ground Truth 越遠的 reward 影響不大。

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

由於我們是根據 greedy 的方式選擇 action，又我們在訓練過程中並沒有精確知道所有 state 的 value，所以有可能出現當下是最佳結果，但並不是最佳解的情況，故我們需要使用 epsilon 來偶爾 random 選擇 action。

9. Explain the necessity of the target network.


target network 是非常重要的，有 target network 能使 training 時的 reward 值變動的更加穩定。

10. Explain the effect of replay buffer size in case of too large or too small.

若 replay buffer 過大，training 過程可能更加穩定，但降低 training 的速度。
若 replay buffer 過小，雖然速度較快，但有可能 overfit，甚至 train 不起來。

Report Bonus

1. Implement and experiment on Double-DQN

$$Y_t^Q = r_{t+1} + \gamma \max_a Q(S_{t+1}, a | \theta^-)$$

$$Y_t^{DoubleQ} = r_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}} Q(S_{t+1}, a | \theta^-) | \theta^-)$$

DDQN 與 DQN 最大的差別在於，Target net 中 action 的選擇，在 DQN 中我們是選擇 Target net 中最大的值，但是這樣有可能會 optimal (過於樂觀)，所以我們改選的 action 為在 Behavior net 中最好的 action (其實還是選擇 target net 裡面的值)

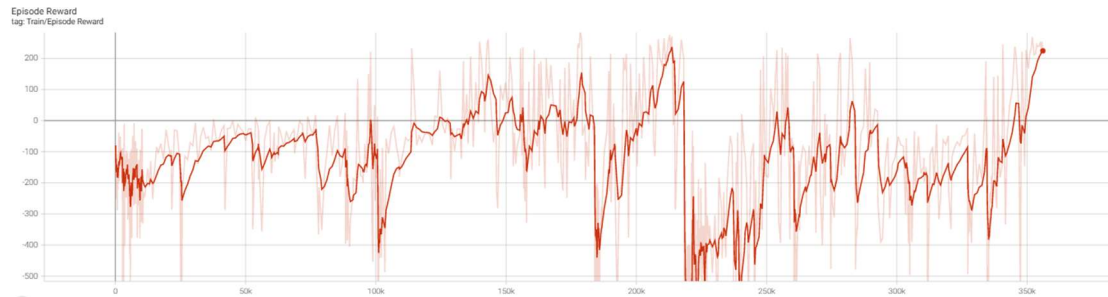
```
def select_action(self, state, epsilon, action_space):
    '''epsilon-greedy based on behavior network'''
    ## TODO ##
    if random.random() < epsilon:
        return action_space.sample()
    else:
        with torch.no_grad():
            return self._behavior_net(torch.tensor(state).to(self.device)).topk(1)[1].item()
```

所以在前面 prep train data 的 select_action 時做出這樣的改動
下為整體表現

```
Start Testing
Episode: 0      Length: 179      Total reward: 254.51
Episode: 1      Length: 186      Total reward: 284.88
Episode: 2      Length: 225      Total reward: 267.44
Episode: 3      Length: 1000     Total reward: 135.91
Episode: 4      Length: 222      Total reward: 311.93
Episode: 5      Length: 198      Total reward: 272.83
Episode: 6      Length: 245      Total reward: 303.42
Episode: 7      Length: 217      Total reward: 290.69
Episode: 8      Length: 257      Total reward: 309.51
Episode: 9      Length: 215      Total reward: 294.80
Average Reward 272.59284376886865
```

2. Extra hyperparameter tuning

另外測試了僅使用 **behavior network**，不使用 **target net** 來進行訓練的部分，可以看到變化幅度非常大，時好時壞並且不容易 **trian** 出好的結果



Performance

1. [LunarLander-v2]

```
Start Testing
Episode: 0    Length: 179    Total reward: 308.00
Episode: 1    Length: 219    Total reward: 294.05
Episode: 2    Length: 213    Total reward: 296.00
Episode: 3    Length: 264    Total reward: 297.83
Episode: 4    Length: 186    Total reward: 270.67
Episode: 5    Length: 187    Total reward: 310.56
Episode: 6    Length: 200    Total reward: 302.17
Episode: 7    Length: 227    Total reward: 308.12
Episode: 8    Length: 181    Total reward: 293.12
Episode: 9    Length: 589    Total reward: 286.06
Average Reward 296.6576463832856
```

2. [LunarLanderContinuous-v2]

```
Start Testing
Episode: 0    Length: 218    Total reward: 298.48
Episode: 1    Length: 264    Total reward: 270.66
Episode: 2    Length: 387    Total reward: 296.93
Episode: 3    Length: 311    Total reward: 263.77
Episode: 4    Length: 396    Total reward: 284.35
Episode: 5    Length: 200    Total reward: 276.98
Episode: 6    Length: 181    Total reward: 264.64
Episode: 7    Length: 162    Total reward: 259.18
Episode: 8    Length: 224    Total reward: 282.78
Episode: 9    Length: 271    Total reward: 256.79
Average Reward 275.455957562109
```