DQN:

```
class Net(nn.Module):
    def __init__(self, state_dim=8, action_dim=4, hidden_dim=128):
        super().__init__()
        ## TODO ##
        self.fc1 = nn.Linear(state_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, action_dim)
        self.relu = nn.ReLU()
        # raise NotImplementedError

    def forward(self, x):
        ## TODO ##
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        return self.fc3(x)
        # raise NotImplementedError
```

behavior 跟 target net，輸入 dimension 為 8 的 state，輸出 dimention 為 4 的 action(No-op, Fire left engine, Fire main engine, Fire right engine)

```
def select_action(self, state, epsilon, action_space):
    '''epsilon-greedy based on behavior network'''
     ## TODO ##
    if np.random.rand() <= epsilon:
        return action_space.sample()

    state = torch.from_numpy(state)
    state = state.to(self.args.device)
    Q_values = self._behavior_net(state)

    return torch.argmax(Q_values, dim=0).item()
    # raise NotImplementedError
```

random小於 epsilon 就隨機選 action，否則從 Q value 選最高的 action

```
def _update_behavior_network(self, gamma):
    # sample a minibatch of transitions
    state, action, reward, next_state, done = self._memory.sample(
        self.batch_size, self.device)

    ## TODO ##
    q_value = self._behavior_net(state).gather(dim=1, index=action.long())
    with torch.no_grad():
        #Q_arg = self._behavior_net(next_state).max(1)[1]
        #q_next = self._target_net(next_state).gather(1, Q_arg.unsqueeze(1))

        q_next = self._target_net(next_state).max(1)[0].unsqueeze(1)
        q_target = gamma * q_next * (1-done) + reward
    criterion = nn.MSELoss()
    loss = criterion(q_value, q_target)
    # q_value = ?
    # with torch.no_grad():
    #    q_next = ?
    #    q_target = ?
    # criterion = ?
    # loss = criterion(q_value, q_target)
    # raise NotImplementedError
    # optimize
    self._optimizer.zero_grad()
    loss.backward()
    nn.utils.clip_grad_norm_(self._behavior_net.parameters(), 5)
    self._optimizer.step()
```

當前 Q value 為 behavior net Q(S, A)，下一個 Q value 為 target net Q(S_next, A)，action 選可得最高Q value 的。套公式得到 q_target 在跟當前 Q value 算平方差得到 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
```

target net 直接複製 behavior net 的參數來更新

DDPG:

```
class ActorNet(nn.Module):
    def __init__(self, state_dim=8, action_dim=2, hidden_dim=(400, 300)):
        super().__init__()
        ## TODO ##
        h1, h2 = hidden_dim

        self.actor = nn.Sequential(
            nn.Linear(state_dim, h1),
            nn.ReLU(),
            nn.Linear(h1, h2),
            nn.ReLU(),
            nn.Linear(h2, action_dim),
            nn.Tanh(),
        )
#         raise NotImplementedError

    def forward(self, x):
        ## TODO ##
        return self.actor(x)
#         raise NotImplementedError
```

ActorNet 的輸出是決定主、左右引擎的 power 是多少，該 power 介於[-1, 1]，所以 activation function 使用tanh()

```
def select_action(self, state, noise=True):
    '''based on the behavior (actor) network and exploration noise'''
    ## TODO ##
    action = self._actor_net(torch.from_numpy(state).to(self.args.device)).detach().cpu().numpy()
    if noise:
        action += self._action_noise.sample()
    return action
      raise NotImplementedError
```

將 state 丟進 actor net 來選擇 action，再加上高斯雜訊來加強 exploration 的能力

Result:

LunarLander-v2 using DQN:

```
!python dqn.py --test_only --render
```

```
Start Testing
Episode: 0      Length: 238     Total reward: 242.73
Episode: 1      Length: 282     Total reward: 266.54
Episode: 2      Length: 245     Total reward: 272.35
Episode: 3      Length: 251     Total reward: 270.51
Episode: 4      Length: 301     Total reward: 295.78
Episode: 5      Length: 236     Total reward: 264.21
Episode: 6      Length: 307     Total reward: 287.19
Episode: 7      Length: 289     Total reward: 278.69
Episode: 8      Length: 342     Total reward: 297.17
Episode: 9      Length: 222     Total reward: 293.68
Average Reward 276.88613735349526
```

LunarLanderContinuous-v2 using DDPG:

```
!python ddpg.py --test_only --render
```

```
Start Testing
Episode: 0      Length: 147     Total reward: 250.06
Episode: 1      Length: 1000    Total reward: 150.98
Episode: 2      Length: 182     Total reward: 283.67
Episode: 3      Length: 179     Total reward: 276.65
Episode: 4      Length: 525     Total reward: -103.53
Episode: 5      Length: 217     Total reward: 255.53
Episode: 6      Length: 1000    Total reward: -42.78
Episode: 7      Length: 159     Total reward: 257.18
Episode: 8      Length: 1000    Total reward: 140.23
Episode: 9      Length: 251     Total reward: 235.99
Average Reward 170.39754475247432
```