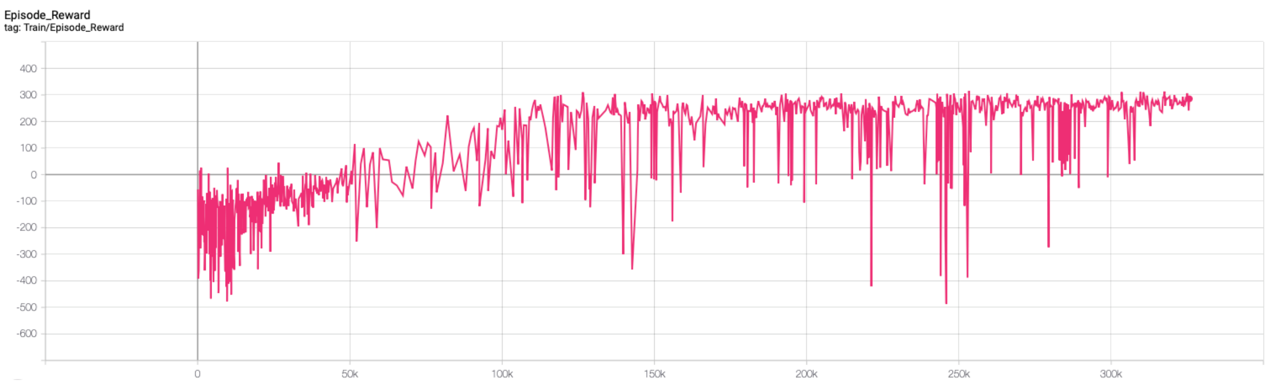
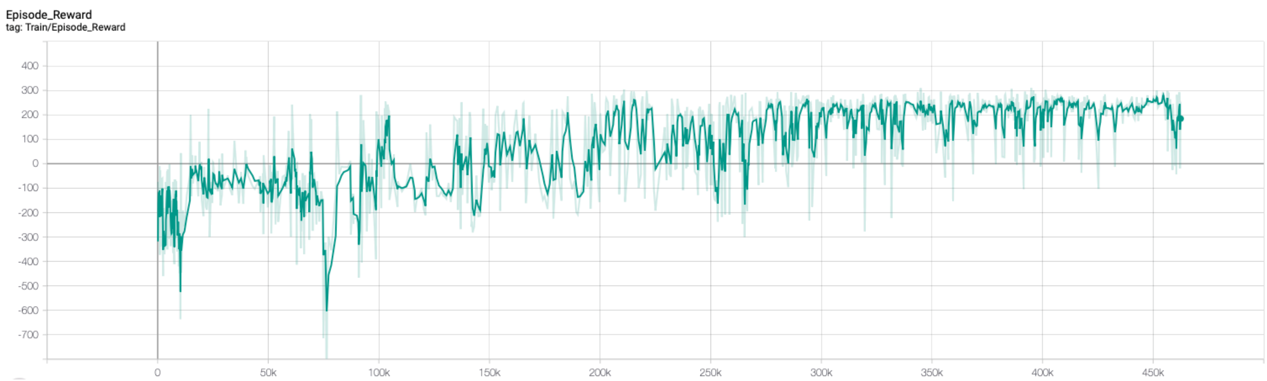
DL HW8

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.
    - Model
        - DQN

There are three fully connected layers in the DQN network. I use the architecture illustrated in spec.

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

    def forward(self, x):
        ## TODO ##
        return self.model(x)
```

- DDPG

I implement the ActorNet. The architecture is the same as that in spec.

```
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()
        )

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

- Optimizer

All optimizer use Adam with corresponding to learning rate.

- DQN

```
        ## TODO ##
        self._optimizer = torch.optim.Adam(self._behavior_net.parameters(), lr=args.lr)
```

- DDPG

```
        ## TODO ##
        self._actor_opt = torch.optim.Adam(self._actor_net.parameters(), lr=args.lra)
        self._critic_opt = torch.optim.Adam(self._critic_net.parameters(), lr=args.lrc)
```

- Select action
- DQN

It uses epsilon-greedy mechanism. We random a number between 0 and 1. If the number is less than epsilon, it's exploration step. We randomly select an action and we can observe the result to judge the performance of network. If the number is greater than epsilon, it's exploitation step. We choose the best action by our network.

```python
def select_action(self, state, epsilon, action_space):
    '''epsilon-greedy based on behavior network'''
    ## TODO ##
    # Exploration
    if np.random.random() < epsilon:
        return action_space.sample()
    # Exploitation
    else:
        actions = self._behavior_net(torch.from_numpy(state).to(self.device))
        return torch.argmax(actions).item()
```

- DDPG

We use actor_net to predict the action. If noise is true, we should add to the result of action prediction.

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

        return action + noise_sample
    else:
        return self._actor_net(torch.from_numpy(state).to(self.device)).detach().cpu().numpy()
```

- Update behavior network
  - DQN

q_value is the Q value we get from behavior network with those chosen action. *q_next* is max Q value from target network. *q_target* is reward add *q_next.* Like the following formula.

$$Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a' | \theta^-)$$

We perform gradient descent on the following formula, so we use mean square error.

$$\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$$

```python
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(1, action.long())
    with torch.no_grad():
        q_next = torch.max(self._target_net(next_state), dim=1)[0].view(-1, 1)
        q_target = reward.view(-1, 1) + gamma * q_next * (1-done)

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

- DDPG

critic_net and *actor_net* should update independently. We update *critic_net* first. *q_value* is computed by *critic_net* with state and chosen action. *a_next* is the action predicted by target *actor_net*. *q_next* is Q value from target network. *q_target* is computed by the formula.

$$y_i = r_i + \gamma Q'\left(s_{t+1}, \mu'\left(s_{t+1} \middle| \theta^{\mu'}\right) \middle| \theta^{Q'}\right)$$

Update the *critic_net* by minimizing the loss.

$$L = \frac{1}{N}\Sigma_i\left(y_i - Q\left(s_i, a_i \middle| \theta^Q\right)\right)^2$$

```
## update critic ##
# critic loss
## TODO ##
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)
```

We update the actor policy using the sampled gradient:

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

```
## update actor ##
# actor loss
## TODO ##
action = self._actor_net(state)
actor_loss = -self._critic_net(state, action).mean()
```

- Update target network
  - DQN

    Load the weights of behavior network to target network.

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

- DDPG

We apply “soft” target updates as the following formula.

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{\mu'}$$

```
@staticmethod
def _update_target_network(target_net, net, tau):
    '''update target network by _soft_ copying from behavior network'''
    for target, behavior in zip(target_net.parameters(), net.parameters()):
        ## TODO ##
        target.data.copy_(tau * behavior.data + (1.0-tau) * target.data)
```

4. Describe differences between your implementation and algorithms. (10%)

All my implementation are similar to the algorithms. It is only different that I modify eps_decay from 0.995 to 0.99998. It make the training process more stable and can have higher score.

5. Describe your implementation and the gradient of actor updating. (10%)

We update the actor policy using the sampled gradient:

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

```
## update actor ##
# actor loss
## TODO ##
action = self._actor_net(state)
actor_loss = -self._critic_net(state, action).mean()
```

6. Describe your implementation and the gradient of critic updating. (10%)

critic_net and *actor_net* should update independently. We update *critic_net* first. *q_value* is computed by *critic_net* with state and chosen action. *a_next* is the action predicted by target *actor_net*. *q_next* is Q value from target network. *q_target* is computed by the formula.

$$y_i = r_i + \gamma Q'\left(s_{t+1}, \mu'\left(s_{t+1}\middle|\theta^{\mu'}\right)\middle|\theta^{Q'}\right)$$

Update the *critic_net* by minimizing the loss.

$$L = \frac{1}{N}\sum_i\left(y_i - Q(s_i, a_i|\theta^{Q})\right)^2$$

```python
## update critic ##
# critic loss
## TODO ##
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)
```

7. Explain effects of the discount factor.

    Take this formula in DQN for example

$$Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a' | \theta^-)$$

Discount factor is gamma in the formula. It can decide the influence of the Q value.

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

    Greedy action selection always select the the best action. Epsilon-greedy has an additional epsilon term in the process. It will random a number. If the number is less than epsilon, it's exploration step. We randomly select an action and we can observe the result to judge the performance of network. If the number is greater than epsilon, it's exploitation step. We choose the best action by our network.

9. Explain the necessity of the target network.

    With this target network, it can make the training more stable. If there is only one network in the training. Consider the following formula. When we update the network, our ground truth target may change too. It's unreasonable, so the target network provide the ground truth concept. It updates lower than behavior network and also use the same weights of behavior network.

$$Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a' | \theta^-)$$

10. Explain the effect of replay buffer size in case of too large or too small. (5%)

If replay buffer size is too big, it waste the memory. If replay buffer size is too small, experience stored may not enough be sampled.

Performance

DQN

Average Reward 283.7794756249046


```
Step: 322728    Episode: 1187   Length: 237    Total reward: 272.16    Ewma reward: 269.42    Epsilon: 0.010
Step: 322972    Episode: 1188   Length: 244    Total reward: 270.76    Ewma reward: 269.49    Epsilon: 0.010
Step: 323219    Episode: 1189   Length: 247    Total reward: 259.77    Ewma reward: 269.00    Epsilon: 0.010
Step: 323534    Episode: 1190   Length: 315    Total reward: 292.54    Ewma reward: 270.18    Epsilon: 0.010
Step: 323781    Episode: 1191   Length: 247    Total reward: 255.37    Ewma reward: 269.44    Epsilon: 0.010
Step: 324001    Episode: 1192   Length: 220    Total reward: 272.18    Ewma reward: 269.58    Epsilon: 0.010
Step: 324255    Episode: 1193   Length: 254    Total reward: 266.68    Ewma reward: 269.43    Epsilon: 0.010
Step: 324447    Episode: 1194   Length: 192    Total reward: 285.19    Ewma reward: 270.22    Epsilon: 0.010
Step: 324656    Episode: 1195   Length: 209    Total reward: 280.66    Ewma reward: 270.74    Epsilon: 0.010
Step: 324933    Episode: 1196   Length: 277    Total reward: 306.31    Ewma reward: 272.52    Epsilon: 0.010
Step: 325221    Episode: 1197   Length: 288    Total reward: 285.17    Ewma reward: 273.15    Epsilon: 0.010
Step: 325446    Episode: 1198   Length: 225    Total reward: 241.85    Ewma reward: 271.59    Epsilon: 0.010
Step: 325679    Episode: 1199   Length: 233    Total reward: 285.50    Ewma reward: 272.28    Epsilon: 0.010
Start Testing
reward: 247.77737925046782
reward: 287.69233449091416
reward: 272.23813057403913
reward: 277.03807451723173
reward: 280.63251297739777
reward: 267.7305995858013
reward: 304.6169023370097
reward: 298.1696910505918
reward: 313.66384964835197
reward: 288.2352818172402
Average Reward 283.7794756249046
```

DDPG

Average Reward 269.7217567932724


```
Step: 456970    Episode: 1185   Length: 171     Total reward: 290.96    Ewma reward: 237.60
Step: 457174    Episode: 1186   Length: 204     Total reward: 232.42    Ewma reward: 237.34
Step: 457752    Episode: 1187   Length: 578     Total reward: 263.28    Ewma reward: 238.64
Step: 458752    Episode: 1188   Length: 1000    Total reward: -25.34    Ewma reward: 225.44
Step: 458982    Episode: 1189   Length: 230     Total reward: 243.93    Ewma reward: 226.36
Step: 459982    Episode: 1190   Length: 1000    Total reward: -8.80     Ewma reward: 214.61
Step: 460260    Episode: 1191   Length: 278     Total reward: 173.78    Ewma reward: 212.56
Step: 460467    Episode: 1192   Length: 207     Total reward: -42.63    Ewma reward: 199.81
Step: 460683    Episode: 1193   Length: 216     Total reward: 281.62    Ewma reward: 203.90
Step: 460941    Episode: 1194   Length: 258     Total reward: 246.39    Ewma reward: 206.02
Step: 461460    Episode: 1195   Length: 519     Total reward: 209.24    Ewma reward: 206.18
Step: 461787    Episode: 1196   Length: 327     Total reward: 292.64    Ewma reward: 210.50
Step: 462016    Episode: 1197   Length: 229     Total reward: 256.46    Ewma reward: 212.80
Step: 462122    Episode: 1198   Length: 106     Total reward: -18.87    Ewma reward: 201.22
Step: 462364    Episode: 1199   Length: 242     Total reward: 253.04    Ewma reward: 203.81
Start Testing
total reward: 246.74
total reward: 279.82
total reward: 275.34
total reward: 260.55
total reward: 281.93
total reward: 270.16
total reward: 273.19
total reward: 288.55
total reward: 297.58
total reward: 223.36
Average Reward 269.7217567932724
```