Deep Q-Network(DQN)

In [2]:
```
import gym

env = gym.make("CartPole-v0", render_mode="human")
```

```
/Users/chohi/project/ai/Reinforcement-Learning/PythonPrj/.venv/lib/pyt
hon3.9/site-packages/gym/envs/registration.py:555: UserWarning: WARN:
The environment CartPole-v0 is out of date. You should consider upgrad
ing to version `v1`.
  logger.warn(
```

In [3]:
```
state = env.reset()[0]
print('상태:', state)

action_space = env.action_space
print('행동의 차원 수:', action_space.n)
```

```
상태: [ 0.02000814 -0.0393481  -0.04406412  0.01341085]
행동의 차원 수: 2
```

In [7]:
```
import gym
import numpy as np

env = gym.make("CartPole-v0", render_mode="human")
state = env.reset()[0]
done = False

while not done:
    env.render()
    action = np.random.choice([0, 1])
    next_state, reward, terminated, truncated, info = env.step(actior
    done = terminated | truncated
env.close()
```

실습 #1 Reply_buffer.py

In [24]:
```
from collections import deque
import random
import numpy as np

class ReplayBuffer:
    def __init__(self, buffer_size, batch_size):
```

```
        self.buffer = deque(maxlen=buffer_size)
        self.batch_size = batch_size

    def add(self, state, action, reward, next_state, done):
        data = (state, action, reward, next_state, done)
        self.buffer.append(data)

    def __len__(self):
        return len(self.buffer)

    def get_batch(self):
        data = random.sample(self.buffer, self.batch_size)

        state = np.stack([x[0] for x in data])
        action = np.array([x[1] for x in data])
        reward = np.array([x[2] for x in data])
        next_state = np.stack([x[3] for x in data])
        done = np.array([x[4] for x in data]).astype(np.int32)
        return state, action, reward, next_state, done
```

In [28]:

```
import gym

env = gym.make("CartPole-v0", render_mode="human")
replay_buffer = ReplayBuffer(buffer_size=10000, batch_size=32)

for episode in range(10):
    state = env.reset()[0]
    done = False

    while not done:
        action = 0

        next_state, reward, terminated, truncated, info = env.step(ac
        done = terminated | truncated

        replay_buffer.add(state, action, reward, next_state, done)
        state = next_state

state, action, reward, next_state, done = replay_buffer.get_batch()

print(state.shape)
print(action.shape)
print(reward.shape)
print(next_state.shape)
print(done.shape)
```

```
(32, 4)
(32,)
(32,)
(32, 4)
(32,)
```

실습 #2 dqn2.py

In [1]:

```
import copy
from collections import deque
import random
import numpy as np
import matplotlib.pyplot as plt
import gym
from dezero import Model
from dezero import optimizers
import dezero.functions as F
import dezero.layers as L

class ReplayBuffer:
    def __init__(self, buffer_size, batch_size):
        self.buffer = deque(maxlen=buffer_size)
        self.batch_size = batch_size

    def add(self, state, action, reward, next_state, done):
        data = (state, action, reward, next_state, done)
        self.buffer.append(data)

    def __len__(self):
        return len(self.buffer)

    def get_batch(self):
        data = random.sample(self.buffer, self.batch_size)

        state = np.stack([x[0] for x in data])
        action = np.array([x[1] for x in data])
        reward = np.array([x[2] for x in data])
        next_state = np.stack([x[3] for x in data])
        done = np.array([x[4] for x in data]).astype(np.int32)
        return state, action, reward, next_state, done
```

In [2]:

```
class QNet(Model):
    def __init__(self, acton_size):
        super().__init__()
        self.l1 = L.Linear(128)
        self.l2 = L.Linear(128)
        self.l3 = L.Linear(acton_size)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = self.l3(x)
        return x
```

In [3]:

```
class DQNAgent:
    def __init__(self):
        self.gamma = 0.98
        self.lr = 0.0005
        self.epsilon = 0.1
```

```
        self.buffer_size = 10000
        self.batch_size = 32
        self.action_size = 2

        self.replay_buffer = ReplayBuffer(self.buffer_size, self.batc
        self.qnet = QNet(self.action_size)
        self.qnet_target = QNet(self.action_size)
        self.optimizer = optimizers.Adam(self.lr)
        self.optimizer.setup(self.qnet)

    def get_action(self, state):
        if np.random.rand() < self.epsilon:
            return np.random.choice(self.action_size)
        else:
            state = state[np.newaxis, :]
            qs = self.qnet(state)
            return qs.data.argmax()

    def update(self, state, action, reward, next_state, done):

        self.replay_buffer.add(state, action, reward, next_state, don
        if len(self.replay_buffer) < self.batch_size:
            return

        state, action, reward, next_state, done = self.replay_buffer.
        qs = self.qnet(state)
        q = qs[np.arange(self.batch_size), action]

        next_qs = self.qnet_target(next_state)
        next_q = next_qs.max(axis=1)
        next_q.unchain()

        target = reward + (1 - done) * self.gamma * next_q
        loss = F.mean_squared_error(q, target)

        self.qnet.cleargrads()
        loss.backward()
        self.optimizer.update()

    def sync_qnet(self):
        self.qnet_target = copy.deepcopy(self.qnet)
```

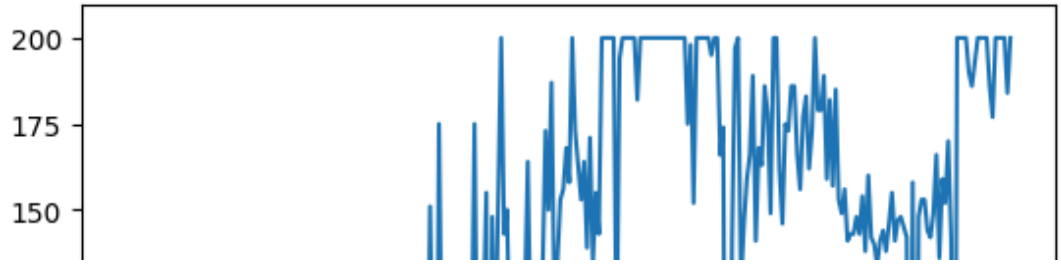
In [4]:

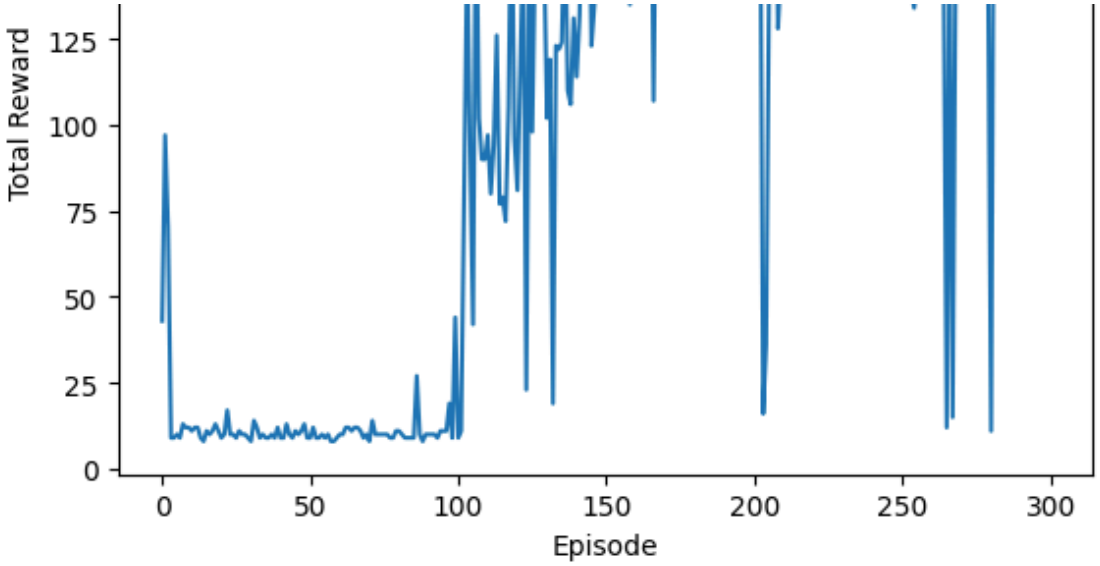
```
episodes = 300
sync_interval = 20
env = gym.make("CartPole-v0", render_mode="rgb_array")
agent = DQNAgent()

reward_history = []

for episode in range(episodes):
    state = env.reset()[0]
    done = False
    total_reward = 0

    while not done:
        action = agent.get_action(state)
        next_state, reward, terminated, truncated, info = env.step(ac
        done = terminated | truncated
```

```
        agent.update(state, action, reward, next_state, done)
        state = next_state
        total_reward += reward

    if episode % sync_interval == 0:
        agent.sync_qnet()

    reward_history.append(total_reward)
    if episode % 10 == 0:
        print("episode :{}, total reward : {}".format(episode, total_

plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.plot(range(len(reward_history)), reward_history)
plt.show()
```

/Users/chohi/project/ai/Reinforcement-Learning/PythonPri/.venv/lib/pyt


```
episode :10, total reward : 11.0
episode :20, total reward : 9.0
episode :30, total reward : 8.0
episode :40, total reward : 9.0
episode :50, total reward : 9.0
episode :60, total reward : 10.0
episode :70, total reward : 8.0
episode :80, total reward : 11.0
episode :90, total reward : 10.0
episode :100, total reward : 9.0
episode :110, total reward : 97.0
episode :120, total reward : 81.0
episode :130, total reward : 102.0
episode :140, total reward : 114.0
episode :150, total reward : 158.0
episode :160, total reward : 143.0
episode :170, total reward : 200.0
episode :180, total reward : 200.0
episode :190, total reward : 175.0
episode :200, total reward : 200.0
episode :210, total reward : 159.0
episode :220, total reward : 200.0
episode :230, total reward : 183.0
episode :240, total reward : 185.0
episode :250, total reward : 138.0
episode :260, total reward : 141.0
episode :270, total reward : 153.0
episode :280, total reward : 11.0
episode :290, total reward : 200.0
```





In [7]:

```
env2 = gym.make("CartPole-v0", render_mode="human")

agent.epsilon = 0
state = env2.reset()[0]
done = False
total_reward = 0

while not done:
    action = agent.get_action(state)
    next_state, reward, terminated, truncated, info = env2.step(actio
    done = terminated | truncated
    state = next_state
    total_reward += reward
    env2.render()
print("total reward : {}".format(total_reward))
```

```
total reward : 200.0
```

In [ ]:

In [8]:

```
##
```