```
pip install gym==0.25.2

    Requirement already satisfied: gym==0.25.2 in /usr/local/lib/python3.10/dist-pac
    Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-p
    Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/d
    Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/d
```

Cart Pole

In this assignment we implement DQN method for the cart pole problem.

Problem Description

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson in “Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem”. A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart. For more information please visit https://www.gymlibrary.dev/environments/classic_control/cart_pole/.

Your Job

1. Read https://www.gymlibrary.dev/environments/classic_control/cart_pole/ to understand the environment, states, reward function, etc.
2. Implement the DQN method.
3. Answer the questions in a pdf.
4. Some helpful API documentation can be found here: https://www.gymlibrary.dev/api/core/.

```
from __future__ import print_function
from __future__ import division
import numpy as np
import gym
from gym import wrappers
import time
import random
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import pandas as pd
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
```

We use environment 'CartPole-v1'. Run the following cell to test the environment.

```
## A demo of operating in this environment with a random policy

def run_random_policy(env, gamma, render = False):
    """
    Run 15 episodes by taking random actions.
    args:
    env: gym environment.
    gamma: discount factor.
    render: boolean to turn rendering on/off.
    """
    obs = env.reset()
    for i in range(15):
        step_idx = 0
        total_reward = 0
        env.reset()
        while True:
            if render:
                env.render()
            #choose an action randomly.
            #env.step() returns a tuple (state, reward, done) where done is a booler
            obs, reward, done , _ ,_= env.step(env.action_space.sample())
            total_reward += (gamma ** step_idx * reward)
            step_idx += 1
            if done:
                break

env_name  = 'CartPole-v1'
env = gym.make(env_name).unwrapped
run_random_policy(env, gamma=0.99, render = True)
env.close()
```

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:49: DeprecationWarning: WARN
If you want to render in human mode, initialize the environment in this way: gym
```

```
  See here for more information: https://www.gymlibrary.ml/content/api/
    deprecation(
```

```
class DoubleDQN(nn.Module):
     #Input: state
     #Output: vector corresponding to state-action values for actions, i.e. Q(s,a).
    def __init__(self):
        super(DoubleDQN, self).__init__()

        ########################### Your Code Here ###########################

        'Define a neural network that takes state as an input and outputs Q(s,a) for
        'You should use ONLY 1 hidden layer with say 50 hidden nodes and ReLU activa
        n_observation = 4
        n_action = 2
        self.layer1 = nn.Linear(n_observation,50)
        self.layer2 = nn.Linear(50,n_action)

        ########################### Your Code Here ###########################

    def forward(self, x):
        ########################### Your Code Here ###########################

        'Define the forward method to get output from your neural network for a give
        x= torch.as_tensor(x).float()
        # print(x, "before Relu")
        x= F.relu(self.layer1(x))
        # print(x,"After Relu")
        return self.layer2(x)

        ########################### Your Code Here ###########################


def choose_action(model, state, EPSILON = 0.9):
    """
    This is the epsilon-greedy strategy.
    """

    ########################### Your Code Here ###########################
    # print("In Choose Action")

    'Write an epsilon greedy strategy to choose action based upon Q(s,a) calculated
    'Return the action'
    if np.random.rand() > EPSILON:
      # print("if here")
      return env.action_space.sample()
    else:
      # print("if there")
      with torch.no_grad():
        return np.argmax(model(state).numpy(), axis=0)

    ########################### Your Code Here ###########################

def copy_parameters(local_model, target_model):
```

```
        """
        Update model parameters.
        local model (PyTorch model): weights will be copied from.
        target model (PyTorch model): weights will be copied to.
        """
        for target_param, local_param in zip(target_model.parameters(), local_model.para
            target_param.data.copy_(local_param.data)


    def plot_returns(returns, x, title):
        '''
        Returns (iterable): list of returns over time
        window: window for rolling mean to smooth plotted curve
        '''
        plt.figure(figsize=(8, 6))  # Adjust figure size if needed
        sns.lineplot(
            data=pd.DataFrame(returns),color=x
        )
        plt.title(title)
        plt.xlabel('Epoch')
        plt.ylabel('Reward')
        plt.show()
```

```
def DoubleDQN_update(model, model_ft, gamma, epochs, render = False):
    """
    Use online learning to update DQN.
    args:
    model (DQN): DQN model
    model_ft (DQN): DQN model for fixed target
    """
    reward_history = []
    initial_reward= []
    optimizer = torch.optim.Adam(model.parameters(), lr = 0.001) #feel free to twea

    #run epoches of episodes
    #stop at done or 10k epochs

    for i in range(epochs):
        # print(i)
        state = env.reset()   #reset enviroment. Get initial state.
        total_reward = 0
        total_reward1 = 0

        if i%100 == 0: #update the fixed target network parameters every 100 epochs
            copy_parameters(model, model_ft)


        ########################### Your Code Here ###########################

        'Train the DQN model'
        'Use SmoothL1Loss() as the loss function'
        'Train it via two methods. '
        '  a) Use the original reward returned by the environment for training'
        '  b) Use the following reward shaping:'
            #x, x_dot, theta, theta_dot = state
            #r1 = (env.x_threshold - abs(x)) / env.x_threshold - 0.8
            #r2 = (env.theta_threshold_radians - abs(theta)) / env.theta_threshold_
            #reward = r1 + r2
        'We will check if there is a difference in convergence rate with and withou

        for t in range(5000):
          action = torch.argmax(model_ft(state)).item()
          # print(action)
          next_state, reward, done, _,_ = env.step(action)
          x, x_dot, theta, theta_dot = next_state
          r1 = (env.x_threshold - abs(x)) / env.x_threshold - 0.8
          r2 = (env.theta_threshold_radians - abs(theta)) / env.theta_threshold_rac
          total_reward1 +=reward
          reward = r1 + r2
          total_reward+=reward
          q_ft= reward+gamma*torch.max(model_ft(state), dim=0)[0]*(1-done)
          criterion = nn.SmoothL1Loss()(torch.max(model(state), dim=0)[0].unsqueeze
        reward_history.append(total_reward)
        initial_reward.append(total_reward1)
```

```
if __name__ == '__main__':
    gamma = 0.99
    np.random.seed(1111)
    env_name  = 'CartPole-v1'
    env = gym.make(env_name).unwrapped
    model = DoubleDQN()
    model_ft = DoubleDQN()
    DoubleDQN_update(model, model_ft, gamma = 0.99, epochs = 200, render = True) #Try
    env.close()

    ## Write any other lines of code if you need it for plotting and other purposes.
```

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning: WAR
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatibility.py:3
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/envs/classic_control/cartpole.py:179
  logger.warn(
```