CartPole Using QLearning

Bhuvi Ghosh
60009210191

```
[ ] import gym
    import numpy as np
    import math
    import matplotlib.pyplot as plt
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `trar
  and should_run_async(code)
```

```
[ ] env = gym.make("CartPole-v0")

    #Environment values
    print(env.observation_space.high) #[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
    print(env.observation_space.low)  #[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
    print(env.action_space.n)     #2
```

```
[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
2
/usr/local/lib/python3.10/dist-packages/gym/envs/registration.py:593: UserWarning: WARN: The environment CartPole-v0 is out c
  logger.warn(
/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning: WARN: Initializing wrapper in old step API which
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning: WARN: Initializing env
  deprecation(
```

```
[ ] #Hyperparamters
    EPISODES = 3000
    DISCOUNT = 0.95
    EPISODE_DISPLAY = 500
    LEARNING_RATE = 0.25
    EPSILON = 0.2
```

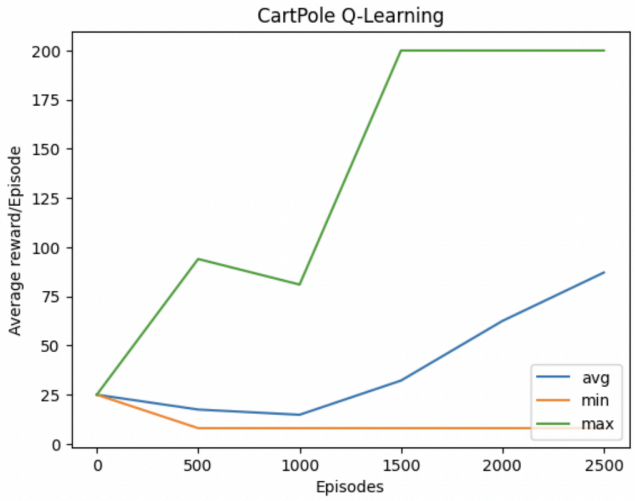
```
[ ] #Q-Table of size theta_state_size*theta_dot_state_size*env.action_space.n
    theta_minmax = env.observation_space.high[2]
    theta_dot_minmax = math.radians(50)
    theta_state_size = 50
    theta_dot_state_size = 50
    Q_TABLE = np.random.randn(theta_state_size,theta_dot_state_size,env.action_space.n)

    # For stats
    ep_rewards = []
    ep_rewards_table = {'ep': [], 'avg': [], 'min': [], 'max': []}
```

```
    def discretised_state(state):
      #state[2] -> theta
      #state[3] -> theta_dot
      discrete_state = np.array([0,0])    #Initialised discrete array

      theta_window =  ( theta_minmax - (-theta_minmax) ) / theta_state_size
      discrete_state[0] = ( state[2] - (-theta_minmax) ) // theta_window
      discrete_state[0] = min(theta_state_size-1, max(0,discrete_state[0]))

      theta_dot_window =  ( theta_dot_minmax - (-theta_dot_minmax) )/ theta_dot_state_size
      discrete_state[1] = ( state[3] - (-theta_dot_minmax) ) // theta_dot_window
      discrete_state[1] = min(theta_dot_state_size-1, max(0,discrete_state[1]))

      return tuple(discrete_state.astype(int))


    for episode in range(EPISODES):
      episode_reward = 0
      curr_discrete_state = discretised_state(env.reset())
      done = False
      i = 0

      if episode % EPISODE_DISPLAY == 0:
        render_state = True
      else:
        render_state = False
```

```
  while not done:
    if np.random.random() > EPSILON:
      action = np.argmax(Q_TABLE[curr_discrete_state])
    else:
      action = np.random.randint(0, env.action_space.n)

    new_state, reward, done, _ = env.step(action)
    new_discrete_state = discretised_state(new_state)
    if render_state:
      env.render()

    if not done:
      max_future_q = np.max(Q_TABLE[new_discrete_state[0],new_discrete_state[1]])
      current_q = Q_TABLE[curr_discrete_state[0],curr_discrete_state[1], action]
      new_q = current_q + LEARNING_RATE*(reward + DISCOUNT*max_future_q - current_q)
      Q_TABLE[curr_discrete_state[0],curr_discrete_state[1], action]=new_q

    i=i+1
    curr_discrete_state = new_discrete_state
    episode_reward += reward

  ep_rewards.append(episode_reward)

  if not episode % EPISODE_DISPLAY:
    avg_reward = sum(ep_rewards[-EPISODE_DISPLAY:])/len(ep_rewards[-EPISODE_DISPLAY:])
    ep_rewards_table['ep'].append(episode)
    ep_rewards_table['avg'].append(avg_reward)
    ep_rewards_table['min'].append(min(ep_rewards[-EPISODE_DISPLAY:]))
    ep_rewards_table['max'].append(max(ep_rewards[-EPISODE_DISPLAY:]))
    print(f"Episode:{episode} avg:{avg_reward} min:{min(ep_rewards[-EPISODE_DISPLAY:])} max:{max(ep_rewards[-EPISODE_DISPLAY:

env.close()
```

```
/usr/local/lib/python3.10/dist-packages/gym/utils/passive_env_checker.py:241: DeprecationWarning: `np.bool8` is a deprecated
  if not isinstance(terminated, (bool, np.bool8)):
/usr/local/lib/python3.10/dist-packages/gym/core.py:49: DeprecationWarning: WARN: You are calling render method, but you didn
If you want to render in human mode, initialize the environment in this way: gym.make('EnvName', render_mode='human') and don
See here for more information: https://www.gymlibrary.ml/content/api/
  deprecation(
Episode:2000 avg:62.508 min:8.0 max:200.0
Episode:2500 avg:87.102 min:8.0 max:200.0
```

```
plt.plot(ep_rewards_table['ep'], ep_rewards_table['avg'], label="avg")
plt.plot(ep_rewards_table['ep'], ep_rewards_table['min'], label="min")
plt.plot(ep_rewards_table['ep'], ep_rewards_table['max'], label="max")
plt.legend(loc=4) #bottom right
plt.title('CartPole Q-Learning')
plt.ylabel('Average reward/Episode')
plt.xlabel('Episodes')
plt.show()
```



CartPole Using SARSA

```
import gym
import numpy as np
import math
import matplotlib.pyplot as plt
```

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

#Environment values
print(env.observation_space.high) #[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
print(env.observation_space.low)  #[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
print(env.action_space.n)     #2

#Hyperparamters
EPISODES = 3000
DISCOUNT = 0.95
EPISODE_DISPLAY = 500
LEARNING_RATE = 0.25
EPSILON = 0.2

#Q-Table of size theta_state_size*theta_dot_state_size*env.action_space.n
theta_minmax = env.observation_space.high[2]
theta_dot_minmax = math.radians(50)
theta_state_size = 50
theta_dot_state_size = 50
Q_TABLE = np.random.randn(theta_state_size,theta_dot_state_size,env.action_space.n)

# For stats
ep_rewards = []
ep_rewards_table = {'ep': [], 'avg': [], 'min': [], 'max': []}
```

```
[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
2
```

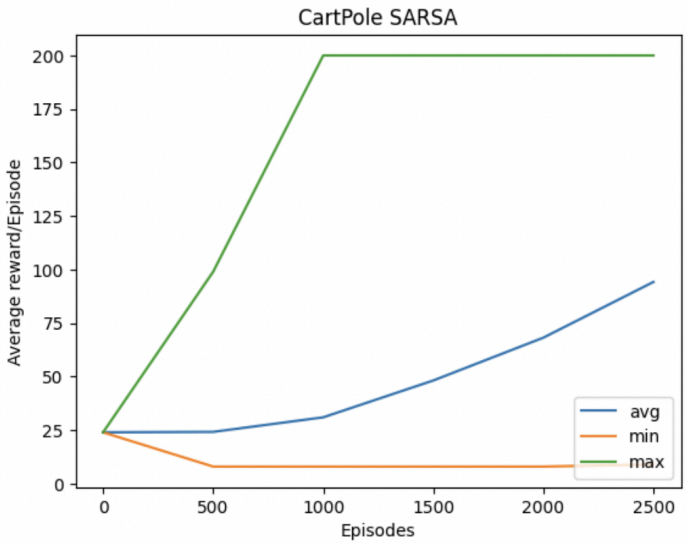
```python
def discretised_state(state):
  #state[2] -> theta
  #state[3] -> theta_dot
  discrete_state = np.array([0,0])    #Initialised discrete array
  theta_window =  ( theta_minmax - (-theta_minmax) ) / theta_state_size
  discrete_state[0] = ( state[2] - (-theta_minmax) ) // theta_window
  discrete_state[0] = min(theta_state_size-1, max(0,discrete_state[0]))

  theta_dot_window =  ( theta_dot_minmax - (-theta_dot_minmax) )/ theta_dot_state_size
  discrete_state[1] = ( state[3] - (-theta_dot_minmax) ) // theta_dot_window
  discrete_state[1] = min(theta_dot_state_size-1, max(0,discrete_state[1]))

  return tuple(discrete_state.astype(int))

for episode in range(EPISODES):
  episode_reward = 0
  done = False

  if episode % EPISODE_DISPLAY == 0:
    render_state = True
  else:
    render_state = False

  curr_discrete_state = discretised_state(env.reset())
  if np.random.random() > EPSILON:
    action = np.argmax(Q_TABLE[curr_discrete_state])
  else:
    action = np.random.randint(0, env.action_space.n)

  while not done:
    new_state, reward, done, _ = env.step(action)
    new_discrete_state = discretised_state(new_state)

    if np.random.random() > EPSILON:
      new_action = np.argmax(Q_TABLE[new_discrete_state])
    else:
      new_action = np.random.randint(0, env.action_space.n)

    if render_state:
      env.render()
```

```
    if not done:
      current_q = Q_TABLE[curr_discrete_state+(action,)]
      max_future_q = Q_TABLE[new_discrete_state+(new_action,)]
      new_q = current_q + LEARNING_RATE*(reward+DISCOUNT*max_future_q-current_q)
      Q_TABLE[curr_discrete_state+(action,)]=new_q

    curr_discrete_state = new_discrete_state
    action = new_action

    episode_reward += reward

  ep_rewards.append(episode_reward)

  if not episode % EPISODE_DISPLAY:
    avg_reward = sum(ep_rewards[-EPISODE_DISPLAY:])/len(ep_rewards[-EPISODE_DISPLAY:])
    ep_rewards_table['ep'].append(episode)
    ep_rewards_table['avg'].append(avg_reward)
    ep_rewards_table['min'].append(min(ep_rewards[-EPISODE_DISPLAY:]))
    ep_rewards_table['max'].append(max(ep_rewards[-EPISODE_DISPLAY:]))
    print(f"Episode:{episode} avg:{avg_reward} min:{min(ep_rewards[-EPISODE_DISPLAY:])} max:{max(ep_rewards[-EPISODE_DISPL

env.close()
```

```
/usr/local/lib/python3.10/dist-packages/gym/utils/passive_env_checker.py:241: DeprecationWarning: `np.bool8` is a deprecat
  if not isinstance(terminated, (bool, np.bool8)):
/usr/local/lib/python3.10/dist-packages/gym/core.py:49: DeprecationWarning: WARN: You are calling render method, but you d
If you want to render in human mode, initialize the environment in this way: gym.make('EnvName', render_mode='human') and
See here for more information: https://www.gymlibrary.ml/content/api/
  deprecation(
Episode:0 avg:24.0 min:24.0 max:24.0
Episode:500 avg:24.206 min:8.0 max:99.0
Episode:1000 avg:31.002 min:8.0 max:200.0
Episode:1500 avg:48.208 min:8.0 max:200.0
Episode:2000 avg:68.2 min:8.0 max:200.0
Episode:2500 avg:94.22 min:9.0 max:200.0
```

```
[ ] plt.plot(ep_rewards_table['ep'], ep_rewards_table['avg'], label="avg")
    plt.plot(ep_rewards_table['ep'], ep_rewards_table['min'], label="min")
    plt.plot(ep_rewards_table['ep'], ep_rewards_table['max'], label="max")
    plt.legend(loc=4) #bottom right
    plt.title('CartPole SARSA')
```

```
plt.plot(ep_rewards_table['ep'], ep_rewards_table['avg'], label="avg")
plt.plot(ep_rewards_table['ep'], ep_rewards_table['min'], label="min")
plt.plot(ep_rewards_table['ep'], ep_rewards_table['max'], label="max")
plt.legend(loc=4) #bottom right
plt.title('CartPole SARSA')
plt.ylabel('Average reward/Episode')
plt.xlabel('Episodes')
plt.show()
```



+ Code + Text