Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №5
по дисциплине
«Методы машинного обучения»
на тему

«Обучение на основе временны’х различий.»

Выполнил:
студент группы ИУ5-22М
Лун Сыхань

Москва — 2024г.

1. Цель лабораторной работы

Ознакомление с базовыми методами обучения с подкреплением на основе временных различий.

2. Задание

На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

1. SARSA
2. Q-обучение
3. Двойное Q-обучение

для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text/Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

3. Ход выполнения работы

```
import gym
import numpy as np
```

```
# 初始化环境
env = gym.make('MountainCar-v0')
```

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning: WARN: Initializing wrapper in old step API which retu
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning: WARN: Initializing environm
  deprecation(
```

SARSA:

```
# 初始化 CartPole 环境
env = gym.make('CartPole-v1')

# 设置分箱的参数
num_buckets = (1, 1, 6, 3)  # 每个状态的分箱数量
state_bounds = list(zip(env.observation_space.low, env.observation_space.high))
state_bounds[1] = [-0.5, 0.5]
state_bounds[3] = [-np.radians(50), np.radians(50)]
buckets = [np.linspace(state_bounds[i][0], state_bounds[i][1], num_buckets[i] - 1) for i in range(len(state_bounds))]

# 离散化状态向量
def discretize_state(state):
    discrete_state = []
    for i in range(len(state)):
        discrete_state.append(np.digitize(state[i], buckets[i]))
    return tuple(discrete_state)

# 初始化 Q 表
Q = np.zeros(num_buckets + (env.action_space.n,))

# SARSA 算法
def sarsa(env, Q, num_episodes=1000, alpha=0.1, gamma=0.99, epsilon=0.1):
    for i in range(num_episodes):
        state = env.reset()
        discrete_state = discretize_state(state)
        action = np.argmax(Q[discrete_state])
        while True:
            if np.random.rand() < epsilon:
                action = env.action_space.sample()
            next_state, reward, done, _ = env.step(action)
            next_discrete_state = discretize_state(next_state)
            next_action = np.argmax(Q[next_discrete_state])
            target = reward + gamma * Q[next_discrete_state][next_action]
            Q[discrete_state][action] += alpha * (target - Q[discrete_state][action])
            state = next_state
            discrete_state = next_discrete_state
            action = next_action
            if done:
                break

# 运行 SARSA 算法
sarsa(env, Q)

# 测试最终策略
state = env.reset()
total_reward = 0
while True:
    env.render()
    discrete_state = discretize_state(state)
    action = np.argmax(Q[discrete_state])
    state, reward, done, _ = env.step(action)
    total_reward += reward
    if done:
        break

print("Total reward:", total_reward)
```

```
/usr/local/lib/python3.10/dist-packages/gym/utils/passive_env_checker.py:241: DeprecationWarning: `np.bool8` is a deprecated alias for `np.bool_`. (Deprecated NumPy 1.24)
  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't specified the argument render_mode at environment ir
If you want to render in human mode, initialize the environment in this way: gym.make('EnvName', render_mode='human') and don't call the render method.
See here for more information: https://www.gymlibrary.ml/content/api/
  deprecation(
Total reward: 203.0
```

Q-обучение:

```
# 初始化 CartPole 环境
env = gym.make('CartPole-v1')

# 设置分箱的参数
num_buckets = (1, 1, 6, 3)  # 每个状态的分箱数量
state_bounds = list(zip(env.observation_space.low, env.observation_space.high))
state_bounds[1] = [-0.5, 0.5]
state_bounds[3] = [-np.radians(50), np.radians(50)]
buckets = [np.linspace(state_bounds[i][0], state_bounds[i][1], num_buckets[i] - 1) for i in range(len(state_bounds))]

# 离散化状态向量
def discretize_state(state):
    discrete_state = []
    for i in range(len(state)):
        discrete_state.append(np.digitize(state[i], buckets[i]))
    return tuple(discrete_state)

# 初始化 Q 表
Q = np.zeros(num_buckets + (env.action_space.n,))

# Q-learning 算法
def q_learning(env, Q, num_episodes=1000, alpha=0.1, gamma=0.99, epsilon=0.1):
    for i in range(num_episodes):
        state = env.reset()
        discrete_state = discretize_state(state)
        while True:
            if np.random.rand() < epsilon:
                action = env.action_space.sample()
            else:
                action = np.argmax(Q[discrete_state])
            next_state, reward, done, _ = env.step(action)
            next_discrete_state = discretize_state(next_state)
            target = reward + gamma * np.max(Q[next_discrete_state])
            Q[discrete_state][action] += alpha * (target - Q[discrete_state][action])
            discrete_state = next_discrete_state
            if done:
                break

# 运行 Q-learning 算法
q_learning(env, Q)

# 测试最终策略
state = env.reset()
total_reward = 0
while True:
    env.render()
    discrete_state = discretize_state(state)
    action = np.argmax(Q[discrete_state])
    state, reward, done, _ = env.step(action)
    total_reward += reward
    if done:
        break

print("Total reward:", total_reward)
```

```
Total reward: 500.0
```

Двойное Q-обучение:

```
# 初始化 CartPole 环境
env = gym.make('CartPole-v1')

# 设置分箱的参数
num_buckets = (1, 1, 6, 3)  # 每个状态的分箱数量
state_bounds = list(zip(env.observation_space.low, env.observation_space.high))
state_bounds[1] = [-0.5, 0.5]
state_bounds[3] = [-np.radians(50), np.radians(50)]
buckets = [np.linspace(state_bounds[i][0], state_bounds[i][1], num_buckets[i] - 1) for i in range(len(state_bounds))]

# 离散化状态向量
def discretize_state(state):
    discrete_state = []
    for i in range(len(state)):
        discrete_state.append(np.digitize(state[i], buckets[i]))
    return tuple(discrete_state)

# 初始化 Q 表
Q1 = np.zeros(num_buckets + (env.action_space.n,))
Q2 = np.zeros(num_buckets + (env.action_space.n,))

# Double Q-learning 算法
def double_q_learning(env, Q1, Q2, num_episodes=1000, alpha=0.1, gamma=0.99, epsilon=0.1):
    for i in range(num_episodes):
        state = env.reset()
        discrete_state = discretize_state(state)
        while True:
            if np.random.rand() < epsilon:
                action = env.action_space.sample()
            else:
                action = np.argmax(Q1[discrete_state] + Q2[discrete_state])
            next_state, reward, done, _ = env.step(action)
            next_discrete_state = discretize_state(next_state)
            if np.random.rand() < 0.5:
                next_action = np.argmax(Q1[next_discrete_state])
                target = reward + gamma * Q2[next_discrete_state][next_action]
                Q1[discrete_state][action] += alpha * (target - Q1[discrete_state][action])
            else:
                next_action = np.argmax(Q2[next_discrete_state])
                target = reward + gamma * Q1[next_discrete_state][next_action]
                Q2[discrete_state][action] += alpha * (target - Q2[discrete_state][action])
            discrete_state = next_discrete_state
            if done:
                break

# 运行 Double Q-learning 算法
double_q_learning(env, Q1, Q2)

# 测试最终策略
state = env.reset()
total_reward = 0
while True:
    env.render()
    discrete_state = discretize_state(state)
    action = np.argmax(Q1[discrete_state] + Q2[discrete_state])
    state, reward, done, _ = env.step(action)
    total_reward += reward
    if done:
        break

print("Total reward:", total_reward)
```

```
Total reward: 16.0
```