МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №__5__
по дисциплине «Методы машинного обучения»

Тема: «Обучение на основе временны’х различий»

ИСПОЛНИТЕЛЬ: Лу Сяои
ФИО

группа ИУ5И-22М
подпись

"1" __Июнь__ 2023 г.

ПРЕПОДАВАТЕЛЬ: ____________________
ФИО

____
подпись

"__"_________2023 г.

Москва - 2023

описание задания

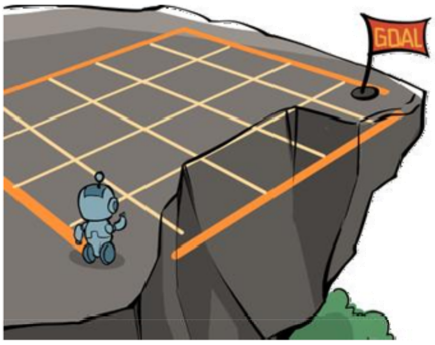
На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

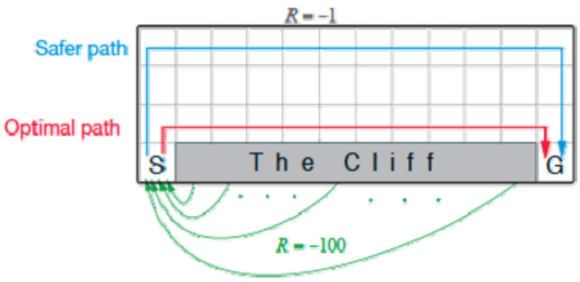
- SARSA
- Q-обучение
- Двойное Q-обучение

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

текст программы и экранные формы с примерами выполнения программы.

Я выбрала среду **Cliff Walking**.





Это мир с сеткой 4x12, где каждая сетка представляет собой состояние. Начальной точкой интеллекта является состояние в левом нижнем углу, а целью - состояние в правом нижнем углу. Интеллект может выполнять 4 действия в каждом состоянии: вверх, вниз, влево и вправо. Если интеллект совершает действие и касается пограничной стены, состояние не меняется, в противном случае он переходит в следующее состояние соответственно. В окружении есть участок обрыва, падение в который или достижение целевого состояния завершает действие и возвращает в начальную точку, т.е. падение в обрыв или достижение целевого состояния является конечным состоянием. Награда за каждый шаг равна -1, награда за падение с обрыва равна -100, а награда за достижение конечного состояния равна 0.

```
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm  # tqdm 是显示循环进度条的库


class CliffWalkingEnv:
    def __init__(self, ncol, nrow):
        self.nrow = nrow
        self.ncol = ncol
        self.x = 0  # 记录当前智能体位置的横坐标
        self.y = self.nrow - 1  # 记录当前智能体位置的纵坐标

    def step(self, action):  # 外部调用这个函数来改变当前位置
        # 4 种动作, change[0]:上, change[1]:下, change[2]:左, change[3]:右。坐标系原点(0,0)
        # 定义在左上角
        change = [[0, -1], [0, 1], [-1, 0], [1, 0]]
        self.x = min(self.ncol - 1, max(0, self.x + change[action][0]))
        self.y = min(self.nrow - 1, max(0, self.y + change[action][1]))
        next_state = self.y * self.ncol + self.x
        reward = -1
        done = False
        if self.y == self.nrow - 1 and self.x > 0:  # 下一个位置在悬崖或者目标
```

```python
            done = True
            if self.x != self.ncol - 1:
                reward = -100
        return next_state, reward, done

    def reset(self):  # 回归初始状态,坐标轴原点在左上角
        self.x = 0
        self.y = self.nrow - 1
        return self.y * self.ncol + self.x
```

- **SARSA**

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

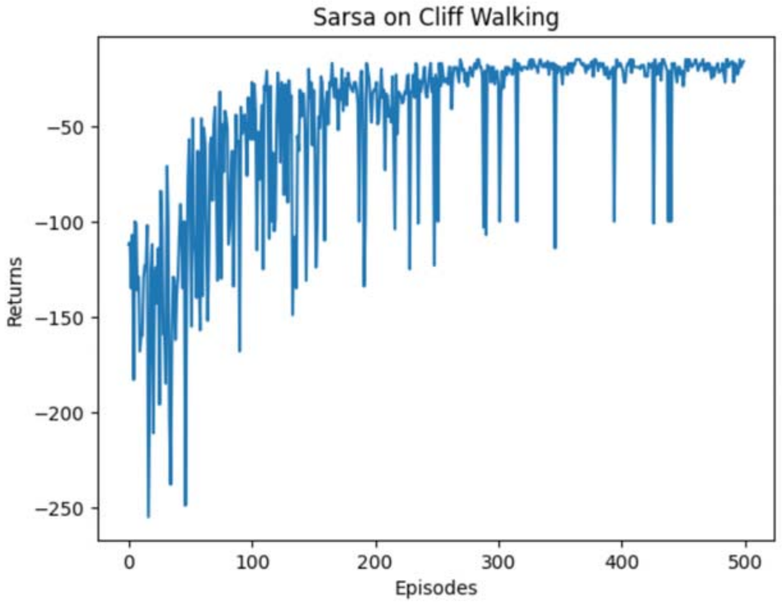

```python
class Sarsa:
    """ Sarsa 算法 """
    def __init__(self, ncol, nrow, epsilon, alpha, gamma, n_action=4):
        self.Q_table = np.zeros([nrow * ncol, n_action])  # 初始化 Q(s,a)表格
        self.n_action = n_action  # 动作个数
        self.alpha = alpha  # 学习率
        self.gamma = gamma  # 折扣因子
        self.epsilon = epsilon  # epsilon-贪婪策略中的参数

    def take_action(self, state):  # 选取下一步的操作,具体实现为 epsilon-贪婪
        if np.random.random() < self.epsilon:
            action = np.random.randint(self.n_action)
        else:
            action = np.argmax(self.Q_table[state])
        return action

    def best_action(self, state):  # 用于打印策略
        Q_max = np.max(self.Q_table[state])
        a = [0 for _ in range(self.n_action)]
        for i in range(self.n_action):  # 若两个动作的价值一样,都会记录下来
            if self.Q_table[state, i] == Q_max:
                a[i] = 1
        return a

    def update(self, s0, a0, r, s1, a1):
        td_error = r + self.gamma * self.Q_table[s1, a1] - self.Q_table[s0, a0]
        self.Q_table[s0, a0] += self.alpha * td_error
```

```python
ncol = 12
nrow = 4
env = CliffWalkingEnv(ncol, nrow)
np.random.seed(0)
epsilon = 0.1
alpha = 0.1
gamma = 0.9
agent = Sarsa(ncol, nrow, epsilon, alpha, gamma)
num_episodes = 500  # 智能体在环境中运行的序列的数量
```

```python
return_list = []  # 记录每一条序列的回报
for i in range(10):  # 显示 10 个进度条
    # tqdm 的进度条功能
    with tqdm(total=int(num_episodes / 10), desc='Iteration %d' % i) as pbar:
        for i_episode in range(int(num_episodes / 10)):  # 每个进度条的序列数
            episode_return = 0
            state = env.reset()
            action = agent.take_action(state)
            done = False
            while not done:
                next_state, reward, done = env.step(action)
                next_action = agent.take_action(next_state)
                episode_return += reward  # 这里回报的计算不进行折扣因子衰减
                agent.update(state, action, reward, next_state, next_action)
                state = next_state
                action = next_action
            return_list.append(episode_return)
            if (i_episode + 1) % 10 == 0:  # 每 10 条序列打印一下这 10 条序列的平均回报
                pbar.set_postfix({
                    'episode':
                    '%d' % (num_episodes / 10 * i + i_episode + 1),
                    'return':
                    '%.3f' % np.mean(return_list[-10:])
                })
            pbar.update(1)
```

```python
episodes_list = list(range(len(return_list)))
plt.plot(episodes_list, return_list)
plt.xlabel('Episodes')
plt.ylabel('Returns')
plt.title('Sarsa on {}'.format('Cliff Walking'))
plt.show()
```

```
Iteration 0: 100%|██████████| 50/50 [00:00<00:00, 358.70it/s, episode=50, return=-119.400]
Iteration 1: 100%|██████████| 50/50 [00:00<00:00, 300.49it/s, episode=100, return=-63.000]
Iteration 2: 100%|██████████| 50/50 [00:00<00:00, 1047.64it/s, episode=150, return=-51.200]
Iteration 3: 100%|██████████| 50/50 [00:00<00:00, 1053.02it/s, episode=200, return=-48.100]
Iteration 4: 100%|██████████| 50/50 [00:00<00:00, 1211.57it/s, episode=250, return=-35.700]
Iteration 5: 100%|██████████| 50/50 [00:00<00:00, 1065.81it/s, episode=300, return=-29.900]
Iteration 6: 100%|██████████| 50/50 [00:00<00:00, 1203.40it/s, episode=350, return=-28.300]
Iteration 7: 100%|██████████| 50/50 [00:00<00:00, 1248.19it/s, episode=400, return=-27.700]
Iteration 8: 100%|██████████| 50/50 [00:00<00:00, 1075.68it/s, episode=450, return=-28.500]
Iteration 9: 100%|██████████| 50/50 [00:00<00:00, 1717.75it/s, episode=500, return=-18.900]
```




```python
def print_agent(agent, env, action_meaning, disaster=[], end=[]):
    for i in range(env.nrow):
```

```python
        for j in range(env.ncol):
            if (i * env.ncol + j) in disaster:
                print('****', end=' ')
            elif (i * env.ncol + j) in end:
                print('EEEE', end=' ')
            else:
                a = agent.best_action(i * env.ncol + j)
                pi_str = ''
                for k in range(len(action_meaning)):
                    pi_str += action_meaning[k] if a[k] > 0 else 'o'
                print(pi_str, end=' ')
        print()


action_meaning = ['^', 'v', '<', '>']
print('The final convergence of Sarsa algorithm yields a strategy of')
print_agent(agent, env, action_meaning, list(range(37, 47)), [47])
```

```
The final convergence of Sarsa algorithm yields a strategy of
ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ovoo
ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ovoo
^ooo ooo> ^ooo ooo> ooo> ooo> ooo> ^ooo ^ooo ooo> ooo> ovoo
^ooo **** **** **** **** **** **** **** **** **** **** EEEE
```

- **Q-обучение**

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

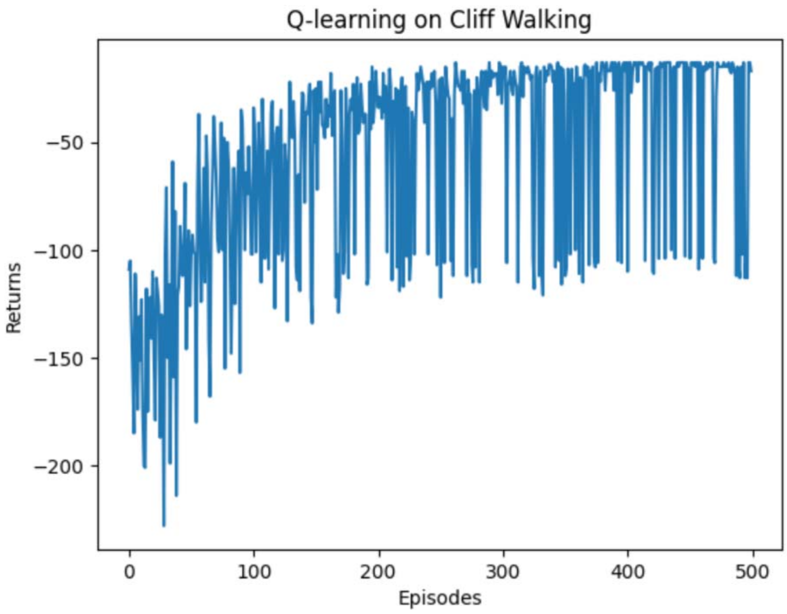

```python
class QLearning:
    """ Q-learning 算法 """
    def __init__(self, ncol, nrow, epsilon, alpha, gamma, n_action=4):
        self.Q_table = np.zeros([nrow * ncol, n_action])  # 初始化Q(s,a)表格
        self.n_action = n_action  # 动作个数
        self.alpha = alpha  # 学习率
        self.gamma = gamma  # 折扣因子
        self.epsilon = epsilon  # epsilon-贪婪策略中的参数

    def take_action(self, state):  #选取下一步的操作
        if np.random.random() < self.epsilon:
            action = np.random.randint(self.n_action)
        else:
            action = np.argmax(self.Q_table[state])
        return action

    def best_action(self, state):  # 用于打印策略
        Q_max = np.max(self.Q_table[state])
        a = [0 for _ in range(self.n_action)]
        for i in range(self.n_action):
            if self.Q_table[state, i] == Q_max:
                a[i] = 1
        return a
```

```python
    def update(self, s0, a0, r, s1):
        td_error = r + self.gamma * self.Q_table[s1].max(
        ) - self.Q_table[s0, a0]
        self.Q_table[s0, a0] += self.alpha * td_error
np.random.seed(0)
epsilon = 0.1
alpha = 0.1
gamma = 0.9
agent = QLearning(ncol, nrow, epsilon, alpha, gamma)
num_episodes = 500  # 智能体在环境中运行的序列的数量

return_list = []  # 记录每一条序列的回报
for i in range(10):  # 显示 10 个进度条
    # tqdm 的进度条功能
    with tqdm(total=int(num_episodes / 10), desc='Iteration %d' % i) as pbar:
        for i_episode in range(int(num_episodes / 10)):  # 每个进度条的序列数
            episode_return = 0
            state = env.reset()
            done = False
            while not done:
                action = agent.take_action(state)
                next_state, reward, done = env.step(action)
                episode_return += reward  # 这里回报的计算不进行折扣因子衰减
                agent.update(state, action, reward, next_state)
                state = next_state
            return_list.append(episode_return)
            if (i_episode + 1) % 10 == 0:  # 每 10 条序列打印一下这 10 条序列的平均回报
                pbar.set_postfix({
                    'episode':
                    '%d' % (num_episodes / 10 * i + i_episode + 1),
                    'return':
                    '%.3f' % np.mean(return_list[-10:])
                })
            pbar.update(1)
```

```python
episodes_list = list(range(len(return_list)))
plt.plot(episodes_list, return_list)
plt.xlabel('Episodes')
plt.ylabel('Returns')
plt.title('Q-learning on {}'.format('Cliff Walking'))
plt.show()
```

```python
action_meaning = ['^', 'v', '<', '>']
print('The final convergence of the Q-learning algorithm yields a strategy of')
print_agent(agent, env, action_meaning, list(range(37, 47)), [47])
```

```
Iteration 0: 100%|██████████| 50/50 [00:00<00:00, 404.74it/s, episode=50, return=-105.700]
Iteration 1: 100%|██████████| 50/50 [00:00<00:00, 286.58it/s, episode=100, return=-70.900]
Iteration 2: 100%|██████████| 50/50 [00:00<00:00, 432.23it/s, episode=150, return=-56.500]
Iteration 3: 100%|██████████| 50/50 [00:00<00:00, 851.78it/s, episode=200, return=-46.500]
Iteration 4: 100%|██████████| 50/50 [00:00<00:00, 718.80it/s, episode=250, return=-40.800]
Iteration 5: 100%|██████████| 50/50 [00:00<00:00, 1058.96it/s, episode=300, return=-20.400]
Iteration 6: 100%|██████████| 50/50 [00:00<00:00, 891.34it/s, episode=350, return=-45.700]
Iteration 7: 100%|██████████| 50/50 [00:00<00:00, 1337.44it/s, episode=400, return=-32.800]
Iteration 8: 100%|██████████| 50/50 [00:00<00:00, 1424.69it/s, episode=450, return=-22.700]
Iteration 9: 100%|██████████| 50/50 [00:00<00:00, 1054.87it/s, episode=500, return=-61.700]
```



The final convergence of the Q-learning algorithm yields a strategy of
^ooo ovoo ovoo ^ooo ^ooo ovoo ooo> ^ooo ^ooo ooo> ooo> ovoo
ooo> ooo> ooo> ooo> ooo> ooo> ^ooo ooo> ooo> ooo> ooo> ovoo
ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ovoo
^ooo **** **** **** **** **** **** **** **** **** **** EEEE

- ## Двойное Q-обучение

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q.(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
    else:
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
    $S \leftarrow S'$
  until S is terminal


```
import gym
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

class DoubleQLearning:
    def __init__(self, env, alpha=0.1, gamma=0.9, epsilon=0.1):
        self.env = env
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.Q1 = np.zeros((env.observation_space.n, env.action_space.n))
        self.Q2 = np.zeros((env.observation_space.n, env.action_space.n))

    def take_action(self, state):
        if np.random.uniform() < self.epsilon:
```

```python
                action = self.env.action_space.sample()
            else:
                action = self.best_action(state)
            return action

    def update(self, state, action, reward, next_state):
        if np.random.uniform() < 0.5:
            Q1, Q2 = self.Q1, self.Q2
        else:
            Q1, Q2 = self.Q2, self.Q1
        a_max = np.argmax(Q1[next_state])
        td_target = reward + self.gamma * Q2[next_state][a_max]
        td_error = td_target - Q1[state][action]
        Q1[state][action] += self.alpha * td_error

    def best_action(self, state):
        return np.argmax(self.Q1[state])

    def train(self, num_episodes=1000):
        return_list = []  # 记录每一条序列的回报
        for i in range(10):  # 显示 10 个进度条
            # tqdm 的进度条功能
            with tqdm(total=int(num_episodes / 10), desc='Iteration %d' % i) as pbar:
                for i_episode in range(int(num_episodes / 10)):  # 每个进度条的序列数
                    episode_return = 0
                    state = self.env.reset()
                    done = False
                    while not done:
                        action = self.take_action(state)
                        next_state, reward, done, info = self.env.step(action)
                        episode_return += reward  # 这里回报的计算不进行折扣因子衰减
                        self.update(state, action, reward, next_state)
                        state = next_state
                    return_list.append(episode_return)
                    if (i_episode + 1) % 10 == 0:  # 每 10 条序列打印一下这 10 条序列的平均回报
                        pbar.set_postfix({
                            'episode':
                            '%d' % (num_episodes / 10 * i + i_episode + 1),
                            'return':
                            '%.3f' % np.mean(return_list[-10:])
                        })
                    pbar.update(1)
        return return_list

    def test(self):
        state = self.env.reset()
        done = False
        steps = 0
        while not done:
            action = self.best_action(state)
            state, reward, done, info = self.env.step(action)
            steps += 1
        return steps
```

```python
env = gym.make('CliffWalking-v0')
agent = DoubleQLearning(env)
return_list = agent.train(num_episodes=500)
```

```python
episodes_list = list(range(len(return_list)))
plt.plot(episodes_list, return_list)
plt.xlabel('Episodes')
plt.ylabel('Returns')
plt.title('Double Q-learning on {}'.format('Cliff Walking'))
plt.show()
```

```
Iteration 0: 100%|██████████| 50/50 [00:00<00:00, 148.59it/s, episode=50, return=-231.300]
Iteration 1: 100%|██████████| 50/50 [00:00<00:00, 333.10it/s, episode=100, return=-45.300]
Iteration 2: 100%|██████████| 50/50 [00:00<00:00, 519.62it/s, episode=150, return=-82.800]
Iteration 3: 100%|██████████| 50/50 [00:00<00:00, 424.74it/s, episode=200, return=-140.500]
Iteration 4: 100%|██████████| 50/50 [00:00<00:00, 250.22it/s, episode=250, return=-30.400]
Iteration 5: 100%|██████████| 50/50 [00:00<00:00, 382.93it/s, episode=300, return=-92.100]
Iteration 6: 100%|██████████| 50/50 [00:00<00:00, 256.39it/s, episode=350, return=-81.900]
Iteration 7: 100%|██████████| 50/50 [00:00<00:00, 411.18it/s, episode=400, return=-71.200]
Iteration 8: 100%|██████████| 50/50 [00:00<00:00, 377.40it/s, episode=450, return=-30.300]
Iteration 9: 100%|██████████| 50/50 [00:00<00:00, 365.72it/s, episode=500, return=-65.800]
```