Homework 1

蔡云龙

21215068

Task 1 Frozen Lake MDP

1. **(coding)** Read through `vi_and_pi.py` and implement `policy_evaluation`, `policy_improvement` and `policy_iteration`. The stopping tolerance (defined as $max_s|V_{old}(s) - V_{new}(s)|$) is tol = 10^{-3}. Use $\gamma = 0.9$. Return the optimal value function and the optimal policy.

   策略迭代算法流程如下:

   Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

   1. Initialization
      $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

   2. Policy Evaluation
      Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
      until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

   3. Policy Improvement
      policy-stable $\leftarrow$ *true*
      For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
      If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

   部分代码:

```
# policy_evaluation function
    ############################
    # YOUR IMPLEMENTATION HERE #
    while True:
        delta = 0.0
        for state in range(nS):
            value, action = value_function[state], policy[state]
            temp_val = 0
            for prob, next_state, reward, done in P[state][action]:
                temp_val += prob * (reward + gamma * value_function[next_state])
            value_function[state] = temp_val
            delta = max(delta, abs(value - value_function[state]))
        if delta < tol:
            break
    ############################

# policy_improvement function
    ############################
    # YOUR IMPLEMENTATION HERE #
    for state in range(nS):
        best_action, best_Q = None, -float('inf')
        for action in range(nA):
            current_Q = 0
            for prob, next_sate, reward, done in P[state][action]:
```

```python
                current_Q += prob * (reward + gamma * value_from_policy[next_sate])
            if current_Q > best_Q:
                best_action, best_Q = action, current_Q
        new_policy[state] = best_action
    ############################

# policy_iteration function
    ############################
    # YOUR IMPLEMENTATION HERE #
    for i in range(100000):
        value_function = policy_evaluation(P, nS, nA, policy, gamma, tol)
        new_policy = policy_improvement(P, nS, nA, value_function, policy, gamma)

        if np.array_equal(new_policy, policy):
            # print('Policy iteration converges at ' + str(n) + ' iterations')
            break
        policy = new_policy
    ############################
```

运行结果

```
-------------------------
Beginning Policy Iteration
-------------------------

SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
Episode reward: 1.000000
```

2. **(coding)** Implement `value_iteration` in `vi_and_pi.py`. The stopping tolerance is tol = 10^{-3}. Use $\gamma = 0.9$. Return the optimal value function and the optimal policy.

值迭代算法流程为:

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$

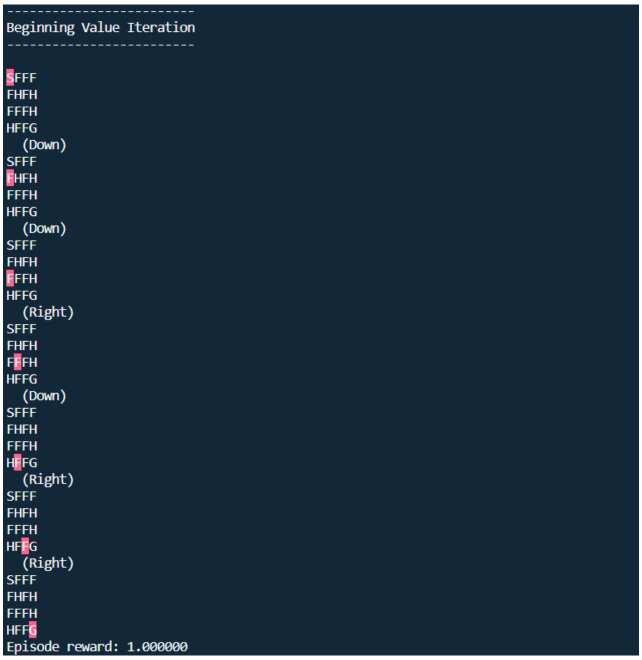
部分代码:

```python
# value_iteration function
    ############################
    # YOUR IMPLEMENTATION HERE #
    for i in range(100000):
        delta = 0.0
        for state in range(nS):
            value = value_function[state]
            best_Q = -float('inf')

            for action in range(nA):
                this_Q = 0
                for (prob, next_state, reward, done) in P[state][action]:
                    this_Q += prob * (reward + gamma * value_function[next_state])
                best_Q = max(best_Q, this_Q)
            value_function[state] = best_Q
            delta = max(delta, abs(value - value_function[state]))
        if delta < tol:
            break

    policy = policy_improvement(P, nS, nA, value_function, policy, gamma)
    ############################
```

运行截图：



Task 2 Test Environment

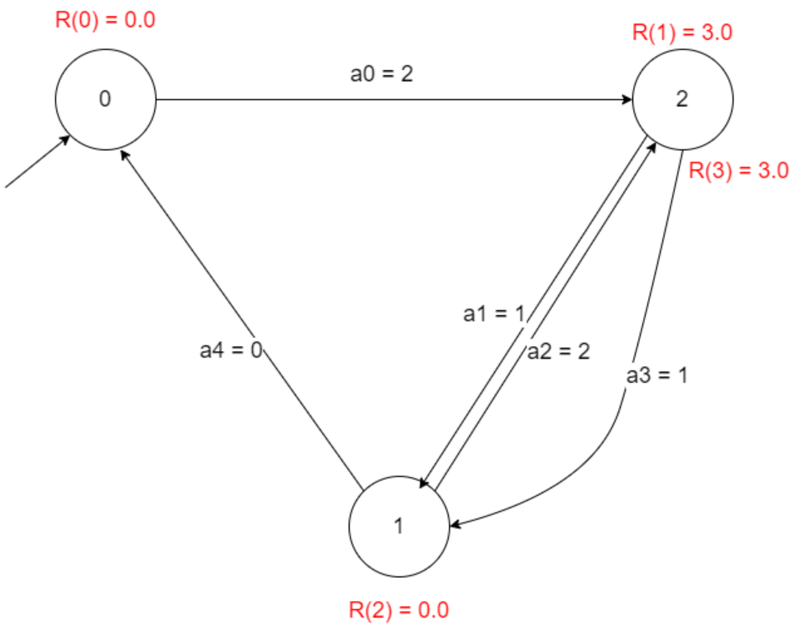
1. **(written)** What is the maximum sum of rewards that can be achieved in a single trajectory in the test environment, assuming γ = 1? Show first that this value is attainable in a single trajectory, and then briefly argue why no other trajectory can achieve greater cumulative reward.

   解：状态过程为：$0 \rightarrow 2 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 0$，即：

   $s_0 = 0, a_0 = 2, R_0 = 0.0, s_1 = 2, a_1 = 1, R_1 = 3.0, s_2 = 1, a_2 = 2, R_2 = 0.0, s_3 = 2, a_3 = 1, R_3 = 3.0, s_4 = 1, a_4 = 0, R_4 = 0.1, s_5 = 0$
   .

   奖励为**6.1**。

   图例如下：



说明：由表格可以看出最大奖励为从2到1，即3；每次执行2->1后要等一次才能再次执行2->1，又一共只有5步，可以执行两次奖励最高的步骤：2->1。一共奖励是6，最后只剩一步，需要从1到0，奖励为0.1，故总的最大奖励为6.1。过程如上所述。

Task 3 Tabular Q-Learning

1. **(coding)** Implement the `get_action` and `update` functions in `q_table.py`. Test your implementation by running `python q_table.py`.

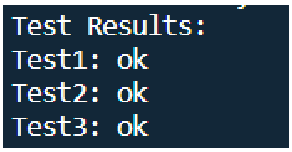
`get_action` 函数中以ϵ 的概率选择一个随机动作，这个过程通过 `env.action_space.sample()` 得到，否则返回参数 `best_action`，故有如下写法：

```
# get_action
        ############################
        # YOUR IMPLEMENTATION HERE #
        if np.random.uniform(0,1) < self.epsilon:
            return self.env.action_space.sample()
        else:
            return best_action
        ############################
```

`Update` 函数中根据当前步数t对ϵ进行线性更新，当t小于总步数时，线性从 `self.eps_begin` 变化到 `self.eps_end`，当t大于总步数时，保持为 `self.eps_end`：

```
# update
        ############################
        # YOUR IMPLEMENTATION HERE #
        if t <= self.nsteps:
            self.epsilon = self.eps_begin - t*(self.eps_begin-self.eps_end)/self.nsteps
        else:
            self.epsilon = self.eps_end
        ############################
```

运行截图：



Task 4 Maze Example

两种算法的流程和对比见下图：

Q-learning vs 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 3
  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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$ 1 A'为下一个state的实际action
    $S \leftarrow S'$; $A \leftarrow A'$; 2
  until S is terminal

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 3
  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'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$ 2
  until S is terminal 1 默认A'为最优策略选的动作

仅`learn`部分代码不同，`check_state_exist`与`choose_action`相同，如下：

```
# __init__
        ############################
        # YOUR IMPLEMENTATION HERE #
        self.q_table = pd.DataFrame(columns=self.actions, dtype=np.float64)
        ############################

# check_state_exist
        ############################
        # YOUR IMPLEMENTATION HERE #
        if state not in self.q_table.index:
            self.q_table = self.q_table.append(
                pd.Series(
                    [0] * len(self.actions),
                    index = self.q_table.columns,
                    name = state,
                )
            )
        ############################
# choose_action
        ############################
        # YOUR IMPLEMENTATION HERE #
        self.check_state_exist(observation)
        # action selection
        if np.random.random() < self.epsilon:
            # choose best action
            state_action = self.q_table.loc[observation, :]

            action = np.random.choice(state_action[state_action == np.max(state_action)].index)
        else:
            # choose random action
            action = np.random.choice(self.actions)
        return action
        ############################
```

由于`SARSA`需要知道下一个动作但`Q Learning`不需要，并且在Q值的更新也不相同，因此两个类的`learn`函数不相同，分别如下：

1. **(coding)** Implement **Sarsa** in `RL_sarsa.py`.

```
    def learn(self, state, action, reward, next_state, next_action):
        ''' update q table '''
        ############################
        # YOUR IMPLEMENTATION HERE #
        self.check_state_exist(next_state)
        q_predict = self.q_table.loc[state, action]
        if next_state != 'terminal':
            q_target = reward + self.gamma * self.q_table.loc[next_state, next_action]  # next state is not terminal
        else:
            q_target = reward  # next state is terminal
        self.q_table.loc[state, action] += self.lr * (q_target - q_predict)  # update
        ############################
```

2. **(coding)** Implement **Q_learning** in `RL_q_learning.py`.

```
    def learn(self, state, action, reward, next_state):
        ''' update q table '''
        ############################
        # YOUR IMPLEMENTATION HERE #
        self.check_state_exist(next_state)
        q_predict = self.q_table.loc[state, action]
        if next_state != 'terminal':
            q_target = reward + self.gamma * self.q_table.loc[next_state, :].max()  # next state is not terminal
        else:
            q_target = reward  # next state is terminal
        self.q_table.loc[state, action] += self.lr * (q_target - q_predict)  # update
        ############################
```

3. 在`run_this.py`文件中，手动导入`argparse`库并添加了一个参数`flag`，用来表示通过哪种方式运行，文件中的`update`函数和`main`函数如下(去注释):

```
def update( flag ):
    for episode in range(100):
        # initial observation
        observation = env.reset()
        if flag == 'SARSA':
            action = RL.choose_action(str(observation))
        while True:
            env.render()

            ############################
            # YOUR IMPLEMENTATION HERE #
            if flag == 'QLearning':
                action = RL.choose_action(str(observation))
            ############################

            # RL take action and get next observation and reward
            observation_, reward, done = env.step(action)

            if flag == 'SARSA':
                action_ = RL.choose_action(str(observation_)) # next action for SARSA

            ############################
            # YOUR IMPLEMENTATION HERE #
            if flag == 'SARSA':
                RL.learn(str(observation), action, reward, str(observation_), action_)
            elif flag == 'QLearning':
                RL.learn(str(observation), action, reward, str(observation_))
            ############################

            # swap observation
            observation = observation_
```

```
            if flag == 'SARSA':
                action = action_
            # break while loop when end of this episode
            if done:
                break
    # end of game
    print('game over')
    env.destroy()

# main函数部分
if __name__ == "__main__":
    env = Maze()
    ############################
    # YOUR IMPLEMENTATION HERE #
    flag = parser.parse_args().flag
    # flag = 1 # 1: SARSA  0: QLearning
    if flag == 'SARSA':
        print("Run with SRASA")
        RL = Sarsa(actions=list(range(env.n_actions)))
    elif flag == 'QLearning':
        print("Run with QLearning")
        RL = QLearning(actions=list(range(env.n_actions)))
    ############################
    env.after(100, update(flag=flag))
    env.mainloop()
```

4. 运行时可以通过 `python run_this.py --flag QLearning` 或 `python run_this.py --flag SARSA` 来指定运行方式，效果如下：