Reinforcement Learning Programming Assignment

Design of Value Iteration Agents and Q-learning Agents

For this assignment, there were two different methods for reinforcement learning that were implemented. The first was a Value Iteration agent, and the second was a Q-learning agent, both on a variant of the Cliff World environment. In this environment, the agent starts in the bottom left corner of the grid, and takes actions that move in any of four directions (up, down, left, right). The agent at the cliff incurs a reward of -100 and the run will terminate, while all other transitions are -1. When the agent reaches the bottom right corner, it'll receive a reward of 10 and the run will terminate.

Value Iteration Agent

The primary objective of the Value Iteration agent is to find the optimal value function, which represents the expected cumulative reward that can be obtained from each state while following the optimal policy. This optimal policy is the result of a convergence of Q-values, calculated using the Bellman Equation, following a number of iterations to reach a stable solution. These Q-values are calculated for non-terminal states by checking each possible action and getting an expected cumulative reward.

```
q_value = sum(
    prob * (self.mdp.getReward(state, action, next_state) + self.discount *
self.values[next_state])
    for next_state, prob in self.mdp.getTransitionStatesAndProbs(state, action)
)
```

The following code is the initialization loop, where the number of iterations determine the efficacy of the convergence.

```
iteration = 0
while iteration < iterations:
    for state in mdp.getStates():
        if not mdp.isTerminal(state):
            action_values = []
            for action in mdp.getPossibleActions(state):
                action_values.append(self.getQValue(state,action))
            self.values[state] = max(action_values)
    iteration += 1
```

ITERATION 1: {(0, 1): -1.0, (0, 0): -1.0, (1, 0): -100.0, (0, 2): -1.0, (1, 1): -1.0, (0, 3): -1.0, (1, 2): -1.0, (1, 3): -1.0, 'TERMINAL_STATE': 0, (2, 1): -1.0, (2, 2): -1.0, (2, 3): -1.0, (2, 0): -100.0, (3, 1): 8.0, (3, 2): 6.2, (3, 3): 4.58, (3, 0): 10.0}

…

ITERATION 3: {(0, 1): -2.71, (0, 0): -2.71, (1, 0): -100.0, (0, 2): -2.71, (1, 1): 4.58, (0, 3): -2.71, (1, 2): 3.122, (1, 3): 1.8098, 'TERMINAL_STATE': 0, (2, 1): 6.2, (2, 2): 4.58, (2, 3): 3.122, (2, 0): -100.0, (3, 1): 8.0, (3, 2): 6.2, (3, 3): 4.58, (3, 0): 10.0}

ITERATION 5: {(0, 1): 3.122, (0, 0): 1.8098, (1, 0): -100.0, (0, 2): 1.8098, (1, 1): 4.58, (0, 3): 0.6288200000000002, (1, 2): 3.122, (1, 3): 1.8098, 'TERMINAL_STATE': 0, (2, 1): 6.2, (2, 2): 4.58, (2, 3): 3.122, (2, 0): -100.0, (3, 1): 8.0, (3, 2): 6.2, (3, 3): 4.58, (3, 0): 10.0}

After each iteration, the Q-values get closer to convergence, and by iteration 5 on this grid, the values have essentially converged. An important point to note is that this method is possible because the grid is known ahead of time, and the start, goal, and bad states are marked out before the attempt at convergence begins.

Once the value estimates have converged, the agent can extract this optimal policy by selecting an action that maximizes the Q-value for each state. This allows the agent to move around this grid and reach the goal state.

Q-learning Agent

As opposed to the Value Iteration agent, which has a model of the environment and knows the probability of going from current state to x and the action cost, the Q-learning agent is for when you don't know anything about the environment. It does this by estimating the quality of taking specific actions in particular states by maintaining a Q-table.

Starting with an empty Q-table, the agent will start interacting with the environment, and in each step, will observe its current state. It will choose whether to explore new actions or exploit its current knowledge based on the exploration strategy which is given by the epsilon value.

```
legal_actions = self.getLegalActions(state)
if not legal_actions:
    return None
# Choose a random action with probability epsilon, else use best policy action
if util.flipCoin(self.epsilon):
    return random.choice(legal_actions)
best_action = self.getPolicy(state)
return best_action
```

Once it has made that action, it will observe that transition reward, and will use what it knows and what it just learned to update the Q-table using the Q-learning algorithm.

```
# Get the Q-value for current state
current_q_value = self.getQValue(state, action)
# Get the maximum Q-value for next state
next_max_q_value = self.getValue(nextState)
# Apply the Q-Learning update rule to compute the new Q-value
# new q = cur Q + alpha(reward for taking action + gamma*max expected reward - cur
Q)
new_q_value = current_q_value + self.alpha*(reward + self.gamma * next_max_q_value -
current_q_value)
self.qValues[(state, action)] = new_q_value
```

Experiment

Value Iteration Agent

The primary test for this agent was to observe the effect that fewer iterations had. To do so, I tested the convergence values after 1, 3, 5, and 10 iterations.

Q-learning Agent

The primary test for this agent was to observe the effect that changing epsilon, the probability of exploration, would have on the Q-table. To do so, I tested at varied epsilon values, all with 10 episodes to allow for proper exploration/exploitation time.

Results

Value Iteration Agent

1 Iteration	3 Iterations
---------- ---------- ---------- -------- -1.00, > -1.00, ^ -1.00, v 4.58, v -1.00, < -1.00, < -1.00, v 6.20, v -1.00, > -1.00, ^ -1.00, > 8.00, v S: -1.00 ^ e: -100.00 e: -100.00 e: 10.00 ---------- ---------- ---------- --------	---------- ---------- ---------- -------- -2.71, ^ 1.81, v 3.12, > 4.58, v -2.71, v 3.12, v 4.58, > 6.20, v -2.71, < 4.58, > 6.20, > 8.00, v S: -2.71 v e: -100.00 e: -100.00 e: 10.00 --------- ---------- ---------- --------
AVERAGE RETURNS FROM START STATE: -6.99810729406002	AVERAGE RETURNS FROM START STATE: -3.7237880781999992

5 Iterations	10 Iterations
--------- ---------- ---------- -------- 0.63, v 1.81, v 3.12, v 4.58, v 1.81, v 3.12, > 4.58, > 6.20, v 3.12, > 4.58, > 6.20, > 8.00, v S: 1.81 ^ e: -100.00 e: -100.00 e: 10.00 --------- ---------- ---------- --------	--------- ---------- ---------- -------- 0.63, v 1.81, > 3.12, > 4.58, v 1.81, v 3.12, > 4.58, > 6.20, v 3.12, > 4.58, > 6.20, > 8.00, v S: 1.81 ^ e: -100.00 e: -100.00 e: 10.00 --------- ---------- ---------- --------
AVERAGE RETURNS FROM START STATE: 1.809800000000001	AVERAGE RETURNS FROM START STATE: 1.809800000000001

Q-learning Agent

ε = 0	ε = 0.3
---------- --------- --------- -------- -1.85, < -1.79, v -1.43, ^ -1.32, v -1.67, v -1.56, v -1.19, > 1.76, v -2.07, > -1.58, > 0.64, > 6.34, v S: -2.60 ^ e: -87.50 e: -75.00 e: 9.69 ---------- --------- --------- --------	---------- --------- --------- -------- -0.88, ^ -0.97, ^ -0.75, v -0.50, < -1.10, ^ -1.09, v -0.75, > -0.20, v -1.46, > -0.84, < -0.07, > 3.62, v S: -1.85 v e: -98.44 e: -50.00 e: 8.75 ---------- --------- --------- --------
AVERAGE RETURNS FROM START STATE: -28.711886229782113	AVERAGE RETURNS FROM START STATE: -42.018623248084694

ε = 0.5	ε = 1.0
---------- --------- --------- -------- -0.50, v -0.97, < -0.50, ^ -0.50, > -0.75, ^ -1.08, < -0.50, v -0.50, ^ -1.46, ^ -0.88, > 0.00, ^ 0.00, ^ S: -1.85 < e: -98.44 e: -87.50 e: 5.00 ---------- --------- --------- --------	---------- --------- ------- ------- 0.00, < 0.00, < 0.00, v 0.00, ^ 0.00, < 0.00, v 0.00, ^ 0.00, v -0.50, > 0.00, < 0.00, < 0.00, ^ S: -0.99 ^ e: -99.80 e: 0.00 e: 5.00 ---------- --------- ------- -------
AVERAGE RETURNS FROM START STATE: -54.99737829376967	AVERAGE RETURNS FROM START STATE: -63.43702515003039

Analysis

Value Iteration Agent

Modifying the number of iterations had a noticeable impact on the convergence of the Q-values, up to a certain extent. Based on the collected data after 5 iterations on this 4x4 grid, the Q-values have almost completely converged, and the average returns from the start state are identical at ~1.8.

However, below 5 iterations, the value iteration agent did not have a model that converged, and therefore would not effectively find the reward state.

Q-learning Agent

Modifying ε had a noticeable impact on the Q-table for this agent, and impacted the average returns from start state for every change in ε. It is also clear that allowing for more exploration (higher ε) is risky, and leads to lower average returns and a Q-table where the agent may run off the cliff.

On the other hand, reducing ε, making the agent have almost no exploration, will lead to a Q-table with slightly worse estimates per state, but a higher average return because the agent will try and go straight toward the reward.