Stats 115 21W **Final Project**

March 10, 2021

In this final project you are going to implement the value iteration algorithm on **page 67 from the book *Reinforcement Learning: An Introduction (Second edition)*** . However, we iterate to update Q table instead of V table.

Settings

In this final project we are using the settings similar to the settings for Figure 17.1 on page 646 in the *Artificial Intelligence: A Modern Approach (Third Edition)* (**But not the same!**).

"A simple 4×3 environment that presents the agent with a sequential decision problem." "The"intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and -1, respectively, and all other states have a reward of -0.04." (*Artificial Intelligence: A Modern Approach (Third Edition)*, P646)

Input information

The information we provide includes:

1) transitionTable. Place the *transitionTable.py* file in the same folder with your main file and run the transitionTable line in the main function. You can use the **viewDictionaryStructure** function to take a look at the transitionTable. The transitionTable is in the format $\{s : \{a : \{s' : P(Result(s,a) = s'|a)\}\}\}$.

Some special states: (1, 1) is a wall. You cannot start from or go to a wall state. (3, 0) and (3, 1) are terminals. You can go to a terminal state, but you cannot only stay in the terminal states if you start from the terminal state.

2) rewardTable. Place the *rewardTable.py* file in the same folder with your main file and run the transitionTable line in the main function. You can use the **viewDictionaryStructure** function to take a look at the rewardTable. This rewardTable is in the format $\{s : \{a : \{s' : R(s,a,s')\}\}\}$.

The rewards are determined by the state you end up in. If you move to a normal state (not wall, not terminal), you will get a reward of -0.04. If you move to the positive terminal state (3, 0), you will get a reward of +1. If you move to the negative terminal state (3, 1), you will get a reward of -1. However, if you start from the terminal state, you will get reward 0.

The rewards we are using is different from the rewards in Figure 17.1, so **your result will not be the same with the example in the book (e. g. Figure 17.3)**.

3) Q. The Q is initialized as 0 for all states and actions. It is in the format of $\{s : \{a : Q(s,a)\}\}$.
4) convergenceTolerance=1×10^{-7}. It represents the θ in the algorithm.
5) roundingTolerance=1×10^{-7}. It is used when you determine the policy from the expected utility. If the difference in expected utility between two actions are smaller than roundingTolerance, the two expected utilities are considered the same.
6) gamma=0.8. It represents the γ in the algorithm.

Task 1 - Update Q

The formula is

$$Q(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} Q(s',a')]$$

,

You are supposed to complete the function **updateQFull** in the *qValueIteration.py* file.

There are five input parameters for the function **updateQFull**. (1) state: s in $Q(s,a)$. (2) action: a in $Q(s,a)$. (3) Q: Q table in $Q(s,a)$. It is a dictionary in the format of $\{s : \{a : Q(s,a)\}\}$. (4) getSPrimeRDistribution: A function taking s and a as input, returning a dictionary. The keys of the dictionary are tuples representing (s',r). The values of the dictionary are probabilities corresponding to the outcome (s',r). (5) gamma: γ in the formula.

The return value of the function **updateQFull** is a scalar, representing the Q value $Q(s,a)$ calculated by the update.

Task 2 - Q Value iteration

You are supposed to complete the function **qValueIteration** in the *qValueIteration.py* file.

There are five input parameters for the function **qValueIteration** . (1) Q: Q table to be updated. It is a dictionary in the format of $\{s : \{a : Q(s,a)\}\}$. (2) updateQ: The function used to update Q table. It has three input arguments: state (s), action (a), and Q table. It returns a scalar $Q(s,a)$. (3) stateSpace: A list of all possible states. (4) actionSpace: A list of all possible actions. (5) convergenceTolerance: A scalar. When the maximum difference between the Q table calculated from one iteration and the Q table calculated from the previous iteration is less than convergenceTolerance, we can say the Q table has converged and stop the iterations.

The return value of the function **qValueIteration** is a Q table, in the format of $\{s : \{a : Q(s,a)\}\}$.

Task 3 - Derive policy from Q table

You are supposed to complete the function **getPolicyFull** in the *qValueIteration.py* file. The policy chooses the optimal action for each state. If there is a tie, choose uniformly randomly from the optimal actions.

There are two input parameters for the function **getPolicyFull**. (1) Q: Q of all actions given a state. It is a dictionary in the format of $\{a : Q(s,a)\}$. (2) roundingTolerance: A scalar. For a fixed state, when the difference between the Q value for two actions is less than roundingTolerance, we can say the Q value for the two actions are the same.

The return value of the function **getPolicyFull** is a dictionary representing the policy. The dictionary is in the format of $\{s : \{a : \pi(a|s)\}\}$. For example, $\{(1,0) : \{(0,1) : 1\}, (2,0) : \{(0,1) : 0.5, (1,0) : 0.5\}\}$

Task 4 - Unit test

You are supposed to develop a unit test for three functions: **getSPrimeRDistributionFull**, **updateQFull**, **getPolicyFull**. Create your own *testQValueIteration_YourLastName_YourFirstName.py* file and use at least two sets of data to test each function.

Submission

Please submit a completed *qValueIteration_YourLastName_YourFirstName.py* file and a *testQValueIteration_YourLastName_YourFirstName.py* file on CCLE before due. **Please submit two seperate files. Do not zip them! The due date and time of the final project is Sunday, 03/21/2021 11:59pm**.