Lab 5: Train Reinforcement Learning Agent in MDP Environment

Abhishek M J - CS21B2018

23-03-2024

MDP Matlab Environment

```
MDP = createMDP(8,["up";"down"]);
```

- This line creates a Markov Decision Process (MDP) object using the createMDP function.
- The first argument, 8, specifies the number of states in the MDP. Imagine eight distinct positions or situations in your environment.
- The second argument, [“up”; “down”], defines the possible actions that can be taken in each state. In this case, you can either move “up” or “down”.

Defining Rewards and Transitions Probabilities

```
% State 1 transition and reward
MDP.T(1,2,1) = 1;
MDP.R(1,2,1) = 3;
MDP.T(1,3,2) = 1;
MDP.R(1,3,2) = 1;
% State 2 transition and reward
MDP.T(2,4,1) = 1;
MDP.R(2,4,1) = 2;
MDP.T(2,5,2) = 1;
MDP.R(2,5,2) = 1;
% State 3 transition and reward
MDP.T(3,5,1) = 1;
MDP.R(3,5,1) = 2;
MDP.T(3,6,2) = 1;
MDP.R(3,6,2) = 4;
% State 4 transition and reward
MDP.T(4,7,1) = 1;
MDP.R(4,7,1) = 3;
MDP.T(4,8,2) = 1;
MDP.R(4,8,2) = 2;
% State 5 transition and reward
MDP.T(5,7,1) = 1;
MDP.R(5,7,1) = 1;
MDP.T(5,8,2) = 1;
MDP.R(5,8,2) = 9;
% State 6 transition and reward
MDP.T(6,7,1) = 1;
MDP.R(6,7,1) = 5;
MDP.T(6,8,2) = 1;
MDP.R(6,8,2) = 1;
```

```
% State 7 transition and reward
MDP.T(7,7,1) = 1;
MDP.R(7,7,1) = 0;
MDP.T(7,7,2) = 1;
MDP.R(7,7,2) = 0;
% State 8 transition and reward
MDP.T(8,8,1) = 1;
MDP.R(8,8,1) = 0;
MDP.T(8,8,2) = 1;
MDP.R(8,8,2) = 0;
```

- This block of code sets the transition and reward probabilities for each state-action pair. For example, MDP.T(1, 2, 2) = 1 sets the probability of transitioning from state 1 to state 2 when action 2 ("down") is taken to 1.
- In simpler terms, the agent moves to down from state 1 it will always (with probability 1) reach state 2.

```
MDP.TerminalStates = ["s7";"s8"];
```

- This line specifies that states 1 and 8 are terminal states. Once the agent reaches one of these states, the episode ends.

```
env = rlMDPEnv(MDP)
```

- This line creates a reinforcement learning environment using the rlMDPEnv function. The environment is based on the MDP object we created earlier.

```
env.ResetFcn = @() 1;
rng(0)
```

- This block of code sets the reset function for the environment to always start from state 1.

Define Q-Table and Initialize Agent

```
obsInfo = getObservationInfo(env);
actInfo = getActionInfo(env);
qTable = rlTable(obsInfo, actInfo);
qFunction = rlQValueFunction(qTable, obsInfo, actInfo);
qOptions = rlOptimizerOptions(LearnRate=1);
```

- This block of code defines the Q-table and Q-value function for the Q-learning agent.
- The rlTable function creates a table to store the Q-values for each state-action pair.
- The rlQValueFunction function creates a Q-value function using the Q-table, observation information, and action information.
- The rlOptimizerOptions function specifies the learning rate for updating the Q-values.

```
agentOpts = rlQAgentOptions;
agentOpts.DiscountFactor = 1;
agentOpts.EpsilonGreedyExploration.Epsilon = 0.9;
agentOpts.EpsilonGreedyExploration.EpsilonDecay = 0.01;
agentOpts.CriticOptimizerOptions = qOptions;
qAgent = rlQAgent(qRepresentation, agentOpts)
```

- This block of code creates a Q-learning agent using the rlQAgent function. The agent uses the Q-value representation and agent options we defined earlier.
- The EpsilonGreedyExploration.EpsilonDecay option is set to 0.01, which means the exploration rate decreases by 0.01 after each episode.
- The exploration rate determines the probability of the agent taking a random action instead of the action with the highest Q-value.

Train the Q-Learning Agent

```
trainOpts = rlTrainingOptions
trainOpts.MaxStepsPerEpisode = 50;
trainOpts.MaxEpisodes = 500;
trainOpts.StopTrainingCriteria = "AverageReward";
trainOpts.StopTrainingValue = 13;
trainOpts.ScoreAveragingWindowLength = 30;
```

- This block of code creates training options using the rlTrainingOptions function. The training options specify the maximum number of steps and episodes, as well as the stopping criteria for training.

```
doTraining = true;
if doTraining
    trainingStats = train(qAgent, env, trainOpts);
else
    load('genericMDPQAgent.mat', 'qAgent')
end
```

- This block of code trains the Q-learning agent using the train function. The training process is controlled by the training options we defined earlier.
- The trainingStats variable stores the training statistics, such as the average reward per episode and the total number of steps taken.

```
Data = sim(qAgent, env)
cumulativeReward = sum(Data.Reward)
```

- This line simulates the Q-learning agent in the environment using the sim function. The Data variable stores the observations, actions, and rewards collected during the simulation.
- The cumulativeReward variable calculates the total reward obtained by the agent during the simulation.

```
QTable = getLearnableParameters(getCritic(qAgent));
QTable{1}
```

- This line retrieves the final Q-table from the Q-learning agent after training.

```
TrueTableValues = [13,12;5,10;11,9;3,2;1,9;5,1;0,0;0,0]
```

- This line defines the true Q-values for each state-action pair in the MDP environment.

```
cumulativeReward =

    13


ans =

  8×2 single matrix

   12.9914    6.4122
    0.4115    9.9948
    9.8946    8.7794
   10.8234   -4.2250
    0.8819    8.9975
    4.3494    3.3191
         0         0
         0         0


TrueTableValues =

    13    12
     5    10
    11     9
     3     2
     1     9
     5     1
     0     0
     0     0
```

Figure 1: Output

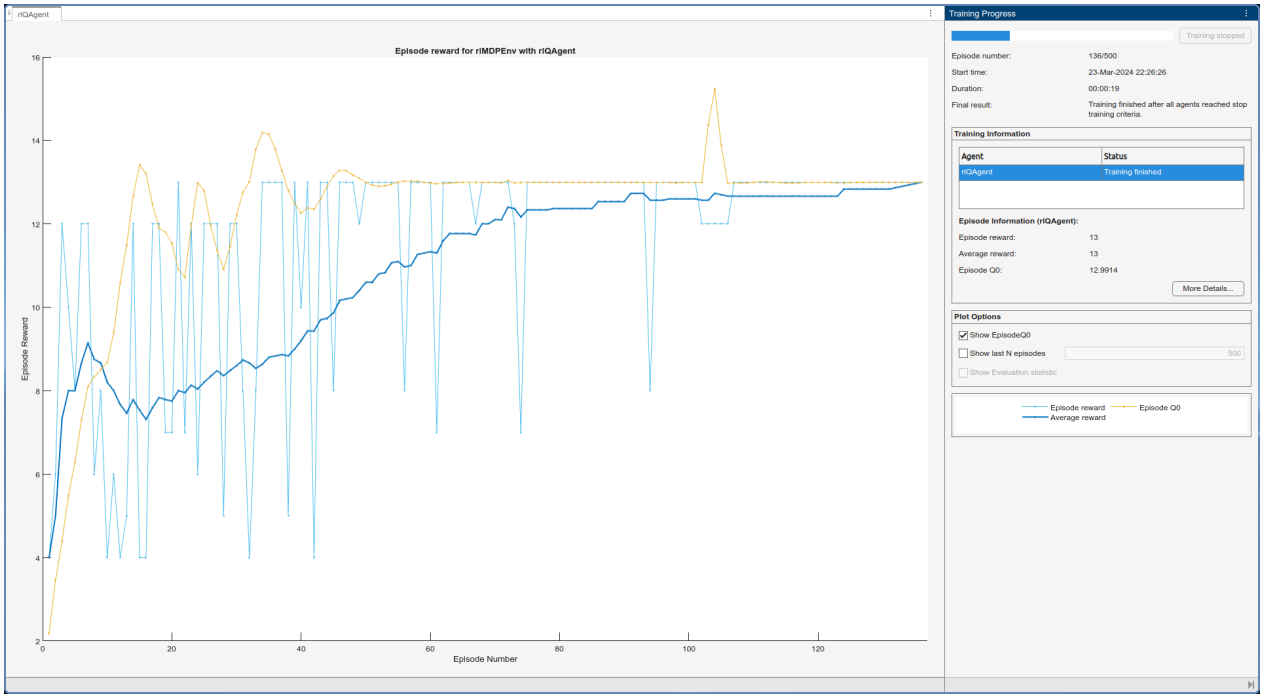

Figure 2: Training Progress