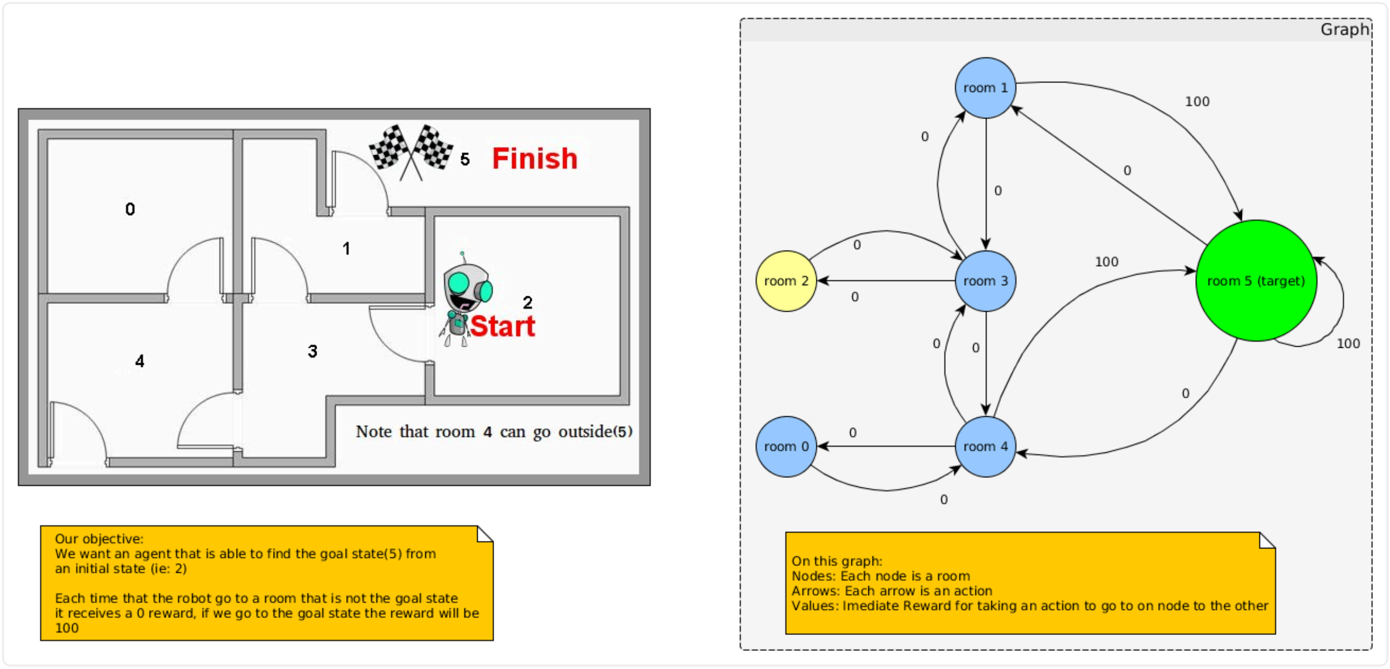
Q_Learning_Simple

Introduction

Q_Learning is a model free reinforcement learning technique. Here we are interested on finding through experiences with the environment the action-value function Q. When the Q function is found we can achieve optimal policy just by selecting the action that gives the biggest expected utility(reward).

Simple Example

Consider a robot that need to learn how to leave a house with the best path possible, on this example we have a house with 5 rooms, and one "exit" room.



Our objective:
We want an agent that is able to find the goal state(5) from an initial state (ie: 2)

Each time that the robot go to a room that is not the goal state it receives a 0 reward, if we go to the goal state the reward will be 100

On this graph:
Nodes: Each node is a room
Arrows: Each arrow is an action
Values: Imediate Reward for taking an action to go to on node to the other

Above we show the house, plant and also a graph representing it. On this graph all rooms are nodes, and the arrows the actions that can be taken on each node. The arrow values, are the immediate rewards that the agent receive by taking some action on a specific room. We choose our reinforcement learning environment to give 0 reward for all rooms that are not the exit room. On our target room we give a 100 reward.

To summarize

- Actions: 0,1,2,3,4,5
- States: 0,1,2,3,4,5
- Rewards:0,100
- Goal state: 5

We can represent all this mechanics on a reward table.

$$
\begin{array}{c} \\ \\ R = \end{array}
\begin{array}{c|rrrrrr}
 & & & \text{Action} & & & \\
\text{State} & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & -1 & -1 & -1 & -1 & 0 & -1 \\
1 & -1 & -1 & -1 & 0 & -1 & 100 \\
2 & -1 & -1 & -1 & 0 & -1 & -1 \\
3 & -1 & 0 & 0 & -1 & 0 & -1 \\
4 & 0 & -1 & -1 & 0 & -1 & 100 \\
5 & -1 & 0 & -1 & -1 & 0 & 100
\end{array}
$$

On this table the rows represent the rooms, and the columns the actions. The values on this matrix represent the rewards, the value (-1) indicate that some specific action is not available.

For example looking the row 4 on this matrix gives the following information:

- Available actions: Go to room 0,3,5
- Possible rewards from room 4: 0 (room 4 to 0),0 (room 4 to 3),100 (room 4 to 5)

The whole point of Q learning is that the matrix R is available only to the environment, the agent need to learn R by himself through experience.

What the agent will have is a Q matrix that encodes, the state,action,rewards, but is initialized with zero, and through experience becomes like the matrix R.

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

As seen on previous chapters, after we have found the Q matrix, we have an optimum policy, and we're done. Basically we just need to use the Q table to choose the action that gives best expected reward. You can also imagine the Q table as the memory of what the agent learned so far through it's experiences.

Algorithm explanation

Just as a quick reminder let's describe the steps on the Q-Learning algorithm

1. Initialize the Q matrix with zeros
2. Select a random initial state
3. For each episode (set of actions that starts on the initial state and ends on the goal state)
   - While state is not goal state
     1. Select a random possible action for the current state
     2. Using this possible action consider going to this next state
     3. Get maximum Q value for this next state (All actions from this next state)
     4. $Q(state, action) = R(state, action) + Gamma * Max[Q(nextState, allActions_{nextState})]$

After we have a good Q table we just need to follow it:

1. Set current state = initial state.
2. From current state, find the action with the highest Q value
3. Set current state = next state(state from action chosen on 2).
4. Repeat Steps 2 and 3 until current state = goal state.

Q-Learning on manual

Let's exercise what we learned so far by doing some episodes by hand. Consider $\gamma = 0.8$ and our initial node(state) to be "room 1"

As we don't have any prior experience we start our Q matrix with zeros

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Now take a look on our reward table (Part of the environment)

$$R = \begin{array}{c|cccccc} & \multicolumn{6}{c}{\text{Action}} \\ \text{State} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

Episode 1

As we start from state "room 1" (second row) there are only the actions 3(reward 0) or 5(reward 100) do be done, imagine that we choose randomly the action 5.

$$
\begin{array}{cc}
 & \text{Action} \\
\text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array}
\end{array}
$$

$$
R = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & \enclose{circle}{0} & -1 & -1 & \enclose{circle}{0} & \enclose{circle}{100}
\end{bmatrix}
$$

On this new (next state 5) there are 3 possible actions: 1 , 4 or 5, with their rewards 0,0,100. Basically is all positive values from row "5", and we're just interested on the one with biggest value. We need to select the biggest Q value with those possible actions by selecting Q(5,1), Q(5,4), Q(5,5), then using a "max" function. But remember that at this state the Q table is still filled with zeros.

$$
Q(1,5) = R(1,5) + 0.8.max([Q(5,1), Q(5,4), Q(5,5)])
$$
$$
\therefore Q(1,5) = 100 + 0.8(0)
$$

As the new state is 5 and this state is the goal state, we finish our episode. Now at the end of this episode the Q matrix will be:

$$
Q = \begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 100 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Episode 2

On this new state we randomly selected the state "room 3", by looking the R matrix we have 3 possible actions on this state, also now by chance we chose the action 1

$$R = \begin{array}{c|cccccc} \text{State} \backslash \text{Action} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & \boxed{0} & \boxed{0} & -1 & \boxed{0} & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

By selecting the action "1" as our next state will have now the following possible actions

$$R = \begin{array}{c|cccccc} \text{State} \backslash \text{Action} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & \boxed{0} & -1 & \boxed{100} \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

Now let's update our Q table:

$$Q(3,1) = R(3,1) + 0.8.max([Q(1,3), Q(1,5)])$$
$$\therefore Q(3,1) = 0 + 0.8(100) = 80$$

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 100 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 80 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

As our new current state 1 is not the goal state, we continue the process. Now by chance from the possible actions of state 1, we choose the action 5. From the action 5 we have the possible actions: 1,4,5 [Q(5,1), Q(5,4), Q(5,5)] unfortunately we did not computed yet this values and our Q matrix remain unchanged.

Episode 100000

After a lot of episodes our Q matrix can be considered to have convergence, on this case Q will be like this:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

if we divide all of the nonzero elements by it's greatest value (on this case 500) we normalize the Q table (Optional):

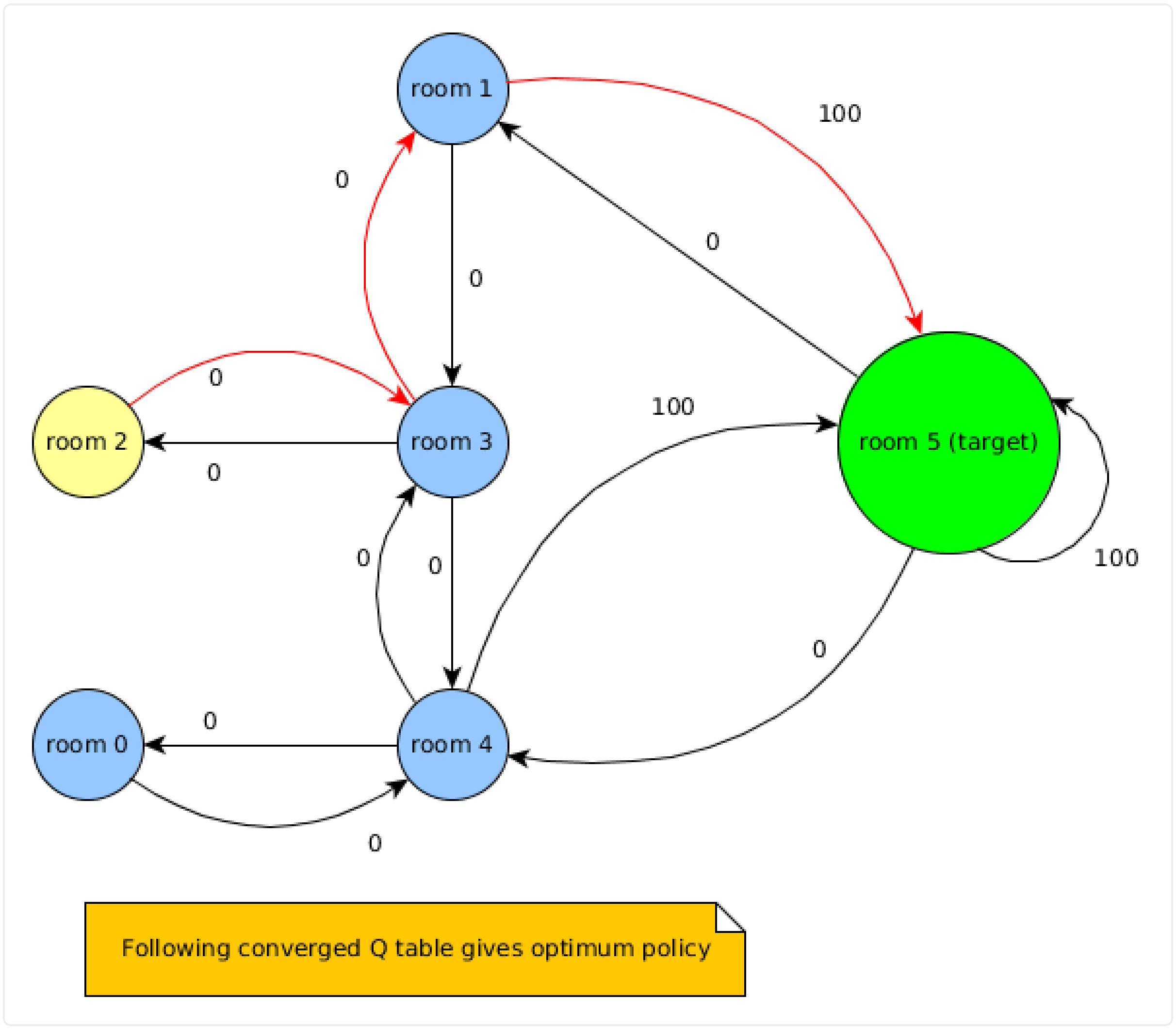
$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

What to do with converged Q table.

Now with the good Q table, imagine that we start from the state "room 2". If we keep choosing the action that gives maximum Q value, we're going from state 2 to 3, then from 3 to 1, then from 1 to 5, and keeps at 5. In other words we choose the actions [2,3,1,5].

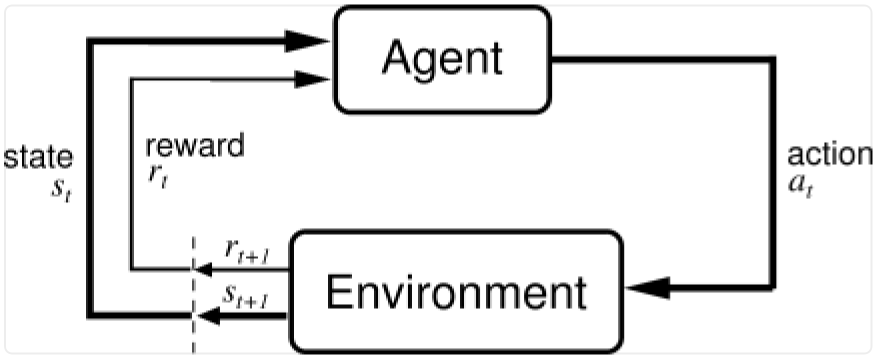
Just one point to pay attention. On state 3 we have the options to choose action 1 or 4, because both have the same max value, we choose the action 1. But the other action would also give the same cumulative reward. [0+0+100]

$$
Q = \begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 0 & 0 & 0 & 80 & 0 \\
1 & 0 & 0 & 0 & 64 & 0 & 100 \\
2 & 0 & 0 & 0 & 64 & 0 & 0 \\
3 & 0 & 80 & 51 & 0 & 80 & 0 \\
4 & 64 & 0 & 0 & 64 & 0 & 100 \\
5 & 0 & 80 & 0 & 0 & 80 & 100
\end{array}
$$



Working out this example in Matlab

Now we're ready to mix those things on the computer, we're going to develop 2 functions, one for representing our agent, and the other for the enviroment.



Enviroment

We start by modeling the environment. Which is the part that receives an action from the agent and give as feedback, a immediate reward and state information. This state is actually the state of the environment after some action is made. Just to remember, if our system is markovian all information necessary for choosing the best future action, is encoded on this state.

Observe that the environment has the matrix R and all models needed to completely implement the universe. For example the environment could be a game, our real world (Grid World) etc...

```
function [ reward, state ] = simple_RL_enviroment( action, restart )
% Simple enviroment of reinforcement learning example
%   http://mnemstudio.org/path-finding-q-learning-tutorial.htm
persistent current_state
if isempty(current_state)
    % Initial random state (excluding goal state)
    current_state = randi([1,5]);
end

% The rows of R encode the states, while the columns encode the action
R = [ -1 -1 -1 -1  0  -1; ...
    -1 -1 -1  0 -1 100; ...
    -1 -1 -1  0 -1  -1; ...
    -1  0  0 -1  0  -1; ...
    0 -1 -1  0 -1 100; ...
    -1  0 -1 -1  0 100 ];

% Sample our R matrix (model)
reward = R(current_state,action);

% Good action taken
if reward ~=-1
    % Returns next state (st+1)
    current_state = action;
end

% Game should be reseted
if restart == true
    % Choose another initial state
    current_state = randi([1,5]);
    reward = -1;
    % We decrement 1 because matlab start the arrays at 1, so just to have
    % the messages with the same value as the figures on the tutorial we
    % take 1....
    fprintf('Enviroment initial state is %d\n',current_state-1);
end

state = current_state;
end
```

Agent

Now on the agent side, we must train to gather experience from the environment. After this we use our learned Q table to act optimally, which means just follow the Q table looking for the actions that gives maximum expected reward. As you may expect the agent interface with the external world through the "simple_RL_enviroment" function.

```
function [ Q ] = simple_RL_agent( )
% Simple agent of reinforcement learning example
%   http://mnemstudio.org/path-finding-q-learning-tutorial.htm
% Train, then normalize Q (divide Q by it's biggest value)
Q = train(); Q = Q / max(Q(:));
% Get the best actions for each possible initial state (1,2,3,4,5)
test(Q);
end

function Q = train()
% Initial training parameters
gamma = 0.8;
goalState=6;
numTrainingEpisodes = 20;
% Set Q initial value
Q = zeros(6,6);

% Learn from enviroment iteraction
for idxEpisode=1:numTrainingEpisodes
    validActionOnState = -1;
    % Reset environment
    [~,currentState] = simple_RL_enviroment(1, true);

    % Episode (initial state to goal state)
    % Break only when we reach the goal state
    while true
        % Choose a random action possible for the current state
        while validActionOnState == -1
            % Select a random possible action
            possibleAction = randi([1,6]);

            % Interact with enviroment and get the immediate reward
            [ reward, ~ ] = simple_RL_enviroment(possibleAction, false);
            validActionOnState = reward;
        end
        validActionOnState = -1;

        % Update Q
        % Get the biggest value from each row of Q, this will create the
        % qMax for each state
        next_state = possibleAction;
        qMax = max(Q,[],2);
        Q(currentState,possibleAction) = reward + ...
            (gamma*(qMax(next_state)));

        if currentState == goalState
            break;
        end
        % Non this simple example the next state will be the action
        currentState = possibleAction;
    end
    fprintf('Finished episode %d restart enviroment\n',idxEpisode);
end
end

function test(Q)
    % Possible permuted initial states, observe that you don't include the
    % goal state 6 (room5)
    possible_initial_states = randperm(5);
    goalState=6;

    % Get the biggest action for every state
    [~, action_max] = max(Q,[],2);

    for idxStates=1:numel(possible_initial_states)
        curr_state = possible_initial_states(idxStates);
        fprintf('initial state room_%d actions=[ ', curr_state-1);
        % Follow optimal policy from intial state to goal state
        while true
            next_state = action_max(curr_state);
            fprintf(' %d,', next_state-1);
            curr_state = next_state;
            if curr_state == goalState
                fprintf(']');
               break
            end
        end
        fprintf('\n');
    end
end
```