```
classdef QLearn < handle
    %QLEARN this class implements a Q-learning agent
    %   This class implementes a Tabular Q-learning agent using both
    properties
        n_s_v; %number of state variable
        n_a; % number of possible actions
        Q; % the Q-table (value function)
        epsilon = 1; % epsilon for epsilon greedy ploicy
        learning_rate = 0.1; % Q-learning learning rate
        discount_factor = 0.999; % Discount factor
        iter; % number of steps of the episode
        e_decay_start; % when epsilon should start decaying
        e_decay_finish; % when epsilon should stop decaying
        e_decay_rate; % how much to decrease epislon during trianing
        state_dimen; % Q-table dimensions
    end

    methods
        function obj = QLearn(num_state_variables,
 state_dimen ,num_actions, epsilon_decay_start, epsilon_decay_finish)
            %QLEARN Construct an instance of this class
            obj.n_s_v = num_state_variables;
            obj.n_a = num_actions;
            obj.state_dimen = state_dimen;
            obj.e_decay_start = epsilon_decay_start;
            obj.e_decay_finish = epsilon_decay_finish;
            obj.e_decay_rate = 1/(obj.e_decay_finish -
 obj.e_decay_start);
            obj.iter = 0;

            % Create Q-table
            obj.Q = zeros(obj.state_dimen(1), obj.state_dimen(2),
 obj.state_dimen(3), obj.n_a);

        end

        function a = action(obj,state)
            %ACTION chooses the action with an epsilon greedy plicy
            obj.iter = obj.iter+1;
            if obj.iter > obj.e_decay_start
                if obj.iter < obj.e_decay_finish
                    obj.epsilon = obj.epsilon - obj.e_decay_rate;
                end
            end

            if rand() < obj.epsilon
                a = randi(obj.n_a);
            else
                max_val = -100000;
                max_action = 1;
                for action = 1:obj.n_a
                    index = num2cell(cat(2, state, action));
```

```
                    val = obj.Q(index{:});
                    if val > max_val
                        max_action = action;
                        max_val = val;
                    end
                end
                a = max_action;
            end
        end

        function a = exploit_action(obj, state)
            %EXPLOIT_ACTION allways chooses the greedy action
            max_val = -100000;
            max_action = 1;
            for action = 1:obj.n_a
                index = num2cell(cat(2, state, action));
                val = obj.Q(index{:});
                if val > max_val
                    max_action = action;
                    max_val = val;
                end
            end
            a = max_action;
        end

        function update_table(obj, S, A, R, Sn)
            %UPDATE_TABLE updates the Q-table value estimates with
 Temporal
            %DFFERENCE LEARNING
            next_best_action = obj.exploit_action(Sn);
            index = num2cell(cat(2,S,A));
            next_index = num2cell(cat(2,Sn,next_best_action));
            obj.Q(index{:}) = obj.Q(index{:}) + obj.learning_rate*(R +
 obj.Q(next_index{:}) - obj.Q(index{:}));
        end
    end
end
```

```
Not enough input arguments.

Error in QLearn (line 21)
            obj.n_s_v = num_state_variables;
```

Published with MATLAB® R2020b