```
classdef DQN < handle
    %DQN This class implemented a DQN algorithm with epsilon greedy
 policy
    properties
        number_state; % number of state variables
        number_actions; % number of possible acitions
        memory; % The memory replay buffer (another object)
        epsilon = 1; % epsilon of the epsilon greedy policy
        epsilon_decay_start; % when to start reducing epsilon
        epsilon_decay_finish; % when to stop reducing epsilon
        epsilon_decay_rate; % how much to decrease epsilon afer
 expreplay
        discount_factor = 0.95; % Discount factor of Q-learning
 equation
        agent; % the agent value function
        epoch = 0; % Number of experience replays
        max_epochs = 10000; % maximum number of expereince replays
        iter; % how many steps of the episode have been taken
    end

    methods
        function obj = DQN(n_S, n_A, layers, e_start, e_finish)
            %DDQN Construct an instance of this class
            obj.number_state = n_S;
            obj.number_actions = n_A;
            obj.memory = Experience_Replay(n_S);
            obj.agent = makenet(layers);
            obj.agent = confignet(obj.agent, rand(n_S, 5), -rand(n_A,
 5));
            obj.agent.trainParam.showWindow=0;
            obj.epsilon_decay_start = e_start;
            obj.epsilon_decay_finish = e_finish;
            obj.epsilon_decay_rate = 1/(e_finish - e_start);
            obj.iter = 0;
        end

        function a = action(obj, state)
            % ACTION choose the action with a greedy policy
            obj.iter = obj.iter +1;
            if obj.iter > obj.epsilon_decay_start
                if obj.iter < obj.epsilon_decay_finish
                    obj.epsilon = obj.epsilon -
 obj.epsilon_decay_rate;
                end
            end
            if rand() < obj.epsilon
                a = randi(obj.number_actions);
            else
                action_vals = obj.agent(transpose(state));
                [~, a] = max(action_vals);
            end
        end
        function a = exploit_action(obj, state)
            % EXPLOIT_ACTION take the greedy action
            action_vals = obj.agent(transpose(state));
            [~, a] = max(action_vals);
        end

        function store(obj, S, A, R, S1)
            % STORE store the expereince in the replay buffer
            obj.memory.insert_experience(S,A,R,S1);
        end

        function experience_replay(obj, batch_size)
           % EXPERIENCE_REPLAY use the Q-learning equation to create
           % targets and train the value function.
           obj.epoch = obj.epoch+1;
           [S, A, R, Sn] = obj.memory.get_batch(batch_size);
           st_predict = obj.agent(transpose(S));
           nst_predict = obj.agent(transpose(Sn));
           Q_target = st_predict;
           for i = 1:batch_size
               [~,next_best_action] = max(nst_predict(:,i));
               Q_target(A(i),i) = R(i) + obj.discount_factor *
 nst_predict(next_best_action,i);
           end
           obj.agent = trainnet(obj.agent, transpose(S), Q_target);
           obj.epoch = obj.epoch + 1;
        end
    end
end


function net = makenet(layers)
    net = feedforwardnet(layers);
end

function net = confignet(net, x, t)
    net = configure(net, x, t);
end

function net = trainnet(net, x, t)
    [net, ~, ~, ~, ~, ~] = adapt(net, x, t);
end
```

```
Not enough input arguments.

Error in DQN (line 21)
            obj.number_state = n_S;
```

Published with MATLAB® R2020b