```
classdef DDQN < handle
    %DDQN Summary of this class goes here
    %   Detailed explanation goes here

    properties
        number_state; % number of state variables
        number_actions; % number of possible actions
        memory; % Replay memory
        epsilon = 1; % epsilon of epsilon greedy policy
        epsilon_decay_start; % when to start decaying epsilon
        epsilon_decay_finish; % when to finishe decaying epsilon
        epsilon_decay_rate; % amount to decrease epsilon
        discount_factor = 0.999; % Q-learning discount factor
        agent; % evaulation nerual network
        target; % target neural network
        epoch = 0; % number of experience replays
        max_epochs = 10000; % Maximum number of expereince replays
        step = 0;

    end

    methods
        function obj = DDQN(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.target = makenet(layers);
            obj.copy_weights_agent_to_target()
            obj.agent = configNet(obj.agent, rand(n_S, 5), -rand(n_A,
 5));
            obj.target = configNet(obj.target, rand(n_S, 5), -
rand(n_A, 5));
            obj.agent.trainParam.showWindow=0;
            obj.target.trainParam.showWindow=0;
            obj.epsilon_decay_start = e_start;
            obj.epsilon_decay_finish = e_finish;
            obj.epsilon_decay_rate = 1/(e_finish - e_start);

        end


        function copy_weights_agent_to_target(obj)
            % Copy the weights of the evaluation netowork to target
 network
             obj.target.IW = obj.agent.IW;
             obj.target.LW = obj.agent.LW;
             obj.target.b = obj.agent.b;
        end
```

```
        function a = action(obj, state)
            % ACTION take an action with the greedy policy
            obj.step = obj.step + 1;
            if obj.step > obj.epsilon_decay_start
                if obj.step < 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 chooses the greedy action
            action_vals = obj.agent(transpose(state));
            [~, a] = max(action_vals);
        end

        function store(obj, S, A, R, S1)
            % STORE store the transition
            obj.memory.insert_experience(S,A,R,S1);
        end

        function experience_replay(obj, batch_size)
            % EXPERINECE _REPLAY Computes the targets using the DDQN
            % Q-learning equation then trains the evaluation netowork
 with
            % the targets
           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));
           nst_predict_target = obj.target(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_target(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 DDQN (line 25)
            obj.number_state = n_S;
```

Published with MATLAB® R2020b