Code for restricted Boltzmann machine task. Notice that I have a fuction at the bottom of the code!

```
% Boltzmann machine written by Axel Qvarnström
clear all
close all
clc

% Inputs
XORInputs = [-1,-1,-1; 1,-1,1; -1,1,1; 1,1,-1]';
allInputs = [-1,-1,-1; 1,-1,1; -1,1,1; 1,1,-1; 1,1,1; 1,-1,-1; -1,-1,1; -1,1,-1]';

% Probability distribution for the data
pData = [0.25, 0.25, 0.25, 0.25, 0, 0, 0, 0];

% Parameters
N = 3;
MValues = [1, 2, 4, 8];
nrOfHiddenNeurons = length(MValues);
nrOfPatterns = length(allInputs);
trials = 1000;
miniBatches = 20;
k = 2000;
eta = 0.005;
nOut = 3000;
nIn = 2000;
dKLSum = zeros(1,nrOfHiddenNeurons);

% Plotting the boundary D_kl
dKL = zeros(1, length(MValues));
for i = 1:length(MValues)
    M = MValues(i);
    if M < 2^(N-1)-1
    dKL(i) = N - log2(M+1) - ((M+1)/(2^(log2(M + 1))));
    else
    dKL(i) = 0;
    end
end
plot(MValues, dKL,'DisplayName','Upper Bound')
hold on

% Running the algorithm for the different M-values
for iHiddenNeuron = 1:nrOfHiddenNeurons
    M = MValues(iHiddenNeuron)

    % Initialize the neurons
    visibleNeurons = zeros(N,1);
    hiddenNeurons = zeros(M,1);
```

```
% Initialize the thresholds
thetaVisible = zeros(N,1);
thetaHidden = zeros(M,1);

% Initialize the weights
weights = normrnd(0, 1, [M N]);
for i = 1:size(weights,1)
        for j = 1:size(weights,2)
            if j == i
                weights(i,i) = 0;      % Making the diagonal weights to zero
            end
        end
end

for itrial = 1:trials
    % Initialize the errors
    deltaWeights = zeros(M,N);
    deltaThetaHidden = zeros(M,1);
    deltaThetaVisible = zeros(N,1);

    for iMiniBatch = 1: miniBatches
        % Pick one pattern randomly from x1-x4
        randomPatternIndex = randi(4);
        feedPattern = XORInputs(:,randomPatternIndex);
        % Initiaize visible neurons as the feed pattern
        visbleNeurons0 = feedPattern;


        % Update hidden neurons,
        localFieldHidden0 = weights * visbleNeurons0 - thetaHidden;
        hiddenNeurons = StochasticUpdate(M,localFieldHidden0);


        for t = 1:k
            % Update visible neurons
            localFieldVisible = weights' * hiddenNeurons - thetaVisible;
            visibleNeurons = StochasticUpdate(N, localFieldVisible);

            % Update hidden neurons
            localFieldHidden = weights * visibleNeurons - thetaHidden;
            hiddenNeurons = StochasticUpdate(M, localFieldHidden);
        end

        % Compute weight and threshold increments
        deltaWeights = deltaWeights + eta*(tanh(localFieldHidden0) * visbleNeurons0' - tanh(l
        deltaThetaHidden = deltaThetaHidden - eta*(tanh(localFieldHidden0) - tanh(localFieldH
        deltaThetaVisible = deltaThetaVisible - eta*(visbleNeurons0 - visibleNeurons);
```

```
        end
        % Updating weight and threshold
        weights = weights + deltaWeights;
        thetaHidden = thetaHidden + deltaThetaHidden;
        thetaVisible = thetaVisible + deltaThetaVisible;

    end

    %% Part when calculating the Kullback-Leibler divergence as a function of the number of hidde
    pB = zeros(1,nrOfPatterns);
    for iOuter = 1:nOut
        randomPatternIndex = randi(nrOfPatterns);
        feedPattern = allInputs(:,randomPatternIndex);
        % Initiaize visible neurons as the feed pattern
        visibleNeurons = feedPattern;

        localFieldHidden = weights * visibleNeurons - thetaHidden;
        hiddenNeurons = StochasticUpdate(M, localFieldHidden);


        for iInner = 1:nIn
            % Update visible neurons
            localFieldVisible = weights' * hiddenNeurons - thetaVisible;
            visibleNeurons = StochasticUpdate(N, localFieldVisible);

            % Update hidden neurons
            localFieldHidden = weights * visibleNeurons - thetaHidden;
            hiddenNeurons = StochasticUpdate(M, localFieldHidden);

            for iPattern = 1:nrOfPatterns
                if visibleNeurons == allInputs(:,iPattern)
                    pB(iPattern) = pB(iPattern) + 1/(nIn * nOut);
                end
            end
        end
    end

    % Calculating the kullback-leiber divergence
    dKL = 0;
    for mu = 1:nrOfPatterns
        if (pData(mu)~=0)
            dKL = dKL + pData(mu) * log(pData(mu)/pB(mu));
        end
    end

    dKLSum(iHiddenNeuron) = dKLSum(iHiddenNeuron) + dKL;
end
```

```
% Plotting the kullback-leiber divergence for the different M-values
plot(MValues,dKLSum,'ro','DisplayName','D_{KL}')
xlabel('Number of hidden neurons [M]')
ylabel('Kullback-leiber divergence [D_{KL}]')


% The function for the stochastic update of the neurons
function neuronValues = StochasticUpdate(nrOfNeurons,localField)

    probability = 1 ./ (1+exp(-2.*localField));
    neuronValues = zeros(nrOfNeurons,1);

    for i = 1:nrOfNeurons
        iProbability = probability(i);
        randomNumber = rand;
        if randomNumber < iProbability
            neuronValues(i) = 1;
        else
            neuronValues(i) = -1;
        end
    end
end
```