```matlab
%ANN HW2 2022 Classification Challenge Machine Nicole Adamah
close
clear
clc
xTest2 = loadmnist2();
[xTrain, tTrain, xVal, tVal, xTest, tTest] = LoadMNIST(3);

%% Neural-network algorithm
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];

options = trainingOptions('sgdm', ...
    'Momentum',0.9,...
    'InitialLearnRate',0.02, ...
    'MaxEpochs',3, ...
    'Shuffle','every-epoch', ...
    'MiniBatchSize',64, ...
    'ValidationData',{xVal tVal}, ...
    'ValidationFrequency',30, ...
    'ValidationPatience',5,...
    'Verbose',false, ...
    'Plots','training-progress');

network = trainNetwork(xTrain,tTrain,layers,options);
```

```
P = classify(network,xTest);
accuracy1 = sum(P == tTest)/numel(tTest);
P_xtest2 = classify(network,xTest2);
%% Print accuracy, plot and save as a csv-file
predicted = (char(P_xtest2));
disp(accuracy1)

n = randperm(10000,20);
for i = 1:20
    subplot(4,5,i);
    colormap(gray(256))
    image(xTest2(:,:,:,n(i)))
    set(gca,'XTick',[], 'YTick', [])
    set(gcf,'Position',[700 700 700 700])
    title("Predicted: " + str2double(predicted(n(i))))
end
writematrix(P_xtest2,"classifications.csv")
```

```matlab
%ANN HW2 2022 One Layer Perceptron Nicole Adamah
close
clear
clc
% Load training and validation data
data = readmatrix('training_set.csv');
val = readmatrix('validation_set.csv');

x1 = normalize(data(:, 1));
x2 = normalize(data(:, 2));
x_inputs = [x1, x2];
t = data(:, 3);

x1_val = normalize(val(:, 1));
x2_val = normalize(val(:, 2));
x_val = [x1_val, x2_val];
t_val = val(:, 3);

% Initializing weights, thresholds and other parameters
M1 = 15;
C = 1;
eta = 0.005;
epochs = 10^3;
w1 = randn([2, M1]);
w2 = randn([1, M1]);
t1 = zeros(1,M1);
t2 = 0;
for epoch = 1:epochs
    for m = 1:length(x_inputs)
    %Iterating over the training set
    mu = randi(length(x_inputs));
    pattern = x_inputs(mu, :);
    % Calculations for hidden layer, V
    V = tanh((pattern * w1) - t1);
    % Calculations for Output
    Output = tanh(sum(w2 * V') - t2);

    % Back propagation in order to update the weights and thresholds
    delta2 = (t(mu) - Output)*(1 - (tanh(dot(w2, V) - t2)^2));
    delta1 = (w2' * delta2) .* (1 - (tanh(pattern * w1 - t1).^2))';
    % Updating the weights and threshholds.
    w2 = w2 + (eta * delta2 * V);
    t2 = t2 - eta * delta2;
    w1 = w1 + eta * (delta1 * pattern)';
```

```
    t1 = t1 - eta * delta1';
    end
    % Iterating over the validation set
    pval = length(val);
    errorcalc = 0;
    for mu = 1:pval
        pattern = x_val(mu, :);
        V = tanh((pattern * w1) - t1);
        Output = tanh(sum(w2 * V') - t2);
        errorcalc = errorcalc + abs(sign(Output) - t_val(mu));
    end
    C = errorcalc/(2*pval);
    disp(epoch);
    disp(C)

    if C < 0.12
        csvwrite('w1.csv', w1');
        csvwrite('w2.csv', w2');
        csvwrite('t1.csv', t1');
        csvwrite('t2.csv', t2);
        break
    end
end
```

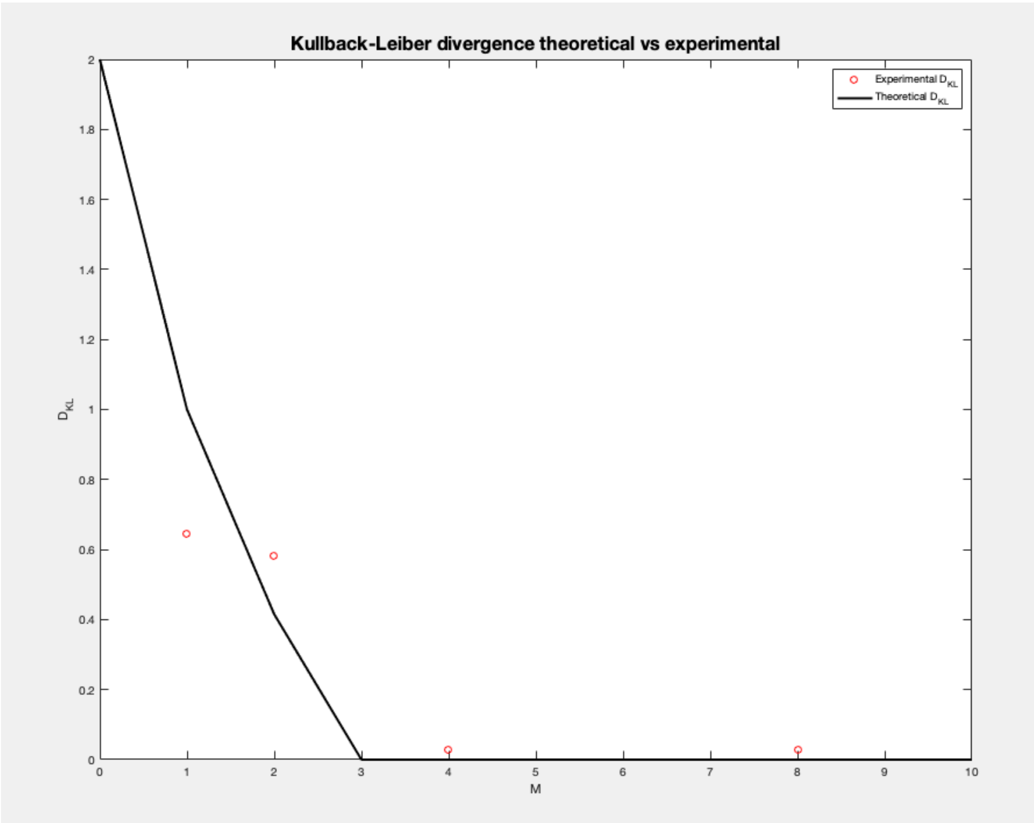
Restricted Boltzmann Machine

In this exercise I have trained a restricted Boltzmann machine to learn the data set of the XOR data. Starting with initializing the weights and thresholds and then training the network where to four specific three-bit patterns, they are assigned probability ¼ and the other patterns are assigned probability zero. The network setup is three visible neurons and M=1,2,4,8 hidden neurons. After the network is trained the Kullback-Leibler divergence is computed as a function of the number of hidden neurons. Finally the experimental values vs the theoretical values of the Kullback-Leibler divergence is plotted. The theoretical value for binary data is obtained from eq 4.40,an upper bound for the Kullback-Leibler divergence:

$$D_{\mathrm{KL}} \leq \log 2 \begin{cases} N - \lfloor \log_2(M+1) \rfloor - \frac{M+1}{2^{\lfloor \log_2(M+1) \rfloor}} & M < 2^{N-1} - 1\,, \\ 0 & M \geq 2^{N-1} - 1\,. \end{cases} \tag{4.40}$$

Results:

Figure 1. Experimental DKL vs Theoretical DKL



In figure one the red dots correspond to the DKL values for different M-values and the black lines are the theoretical DKL, as seen in the figure the Kullback-Leibler divergence goes towards zero. A higher number of hidden neurons results in a better and more precise network as M = 4 and M=8 is very close to zero. Restricted Boltzmann Machines are neural network models characterized by unsupervised learning and specifically only have two layers, visible and hidden. RBM only learns from the input layer which explains why the network performs better with four hidden neurons in comparison to two. In this exercise there are three inputs, so increasing the number of hidden neurons improves the network's representational ability. I found out that the network performed best with eta=0.005 and k=200. When trying higher value on eta the network performed worse and with k=100 and k=400 also made the network perform badly.

```
%ANN HW2 2022 Restricted Boltzmann Machine Nicole Adamah
close
clear
clc

N = 3; %number of visible neurons
M = [1,2,4,8];%number of hidden neurons
eta = 0.005;
inputs = unique(nchoosek(repmat([-1,1], 1, 3), N), 'rows');%combinations for XOR
inputs
P = [1/4 0 0 1/4 0 1/4 1/4 0]';% P(x) = 1/4 for x-inputs: 1, 4, 6, 7.
in = inputs([1, 4, 6, 7],:);
total_x = length(inputs);
minibatches = 20;
k = 200;
trials = 1000;
DKL = zeros(5, length(M));

for Counts = 1:5

for nrneurons=1:length(M)
      % Initilize weights, thresholds and states
      w = normrnd(0,1,[M(nrneurons),N]);
      t_v = zeros(1, N);
      t_h = zeros(M(nrneurons), 1);
      v = zeros(1,N);
      h = zeros(1, M(nrneurons));

      for trial = 1:trials
         dw = zeros(M(nrneurons), N);
         dt_v = zeros(1, N);
         dt_h = zeros(M(nrneurons), 1);
         for i = 1:minibatches
            mu = randi(4);
            v0 = in(mu,:);
            b_0 = (w * v0') - t_h;
            % Updating hidden neurons
            for j = 1:M(nrneurons)
               Pr = 1/(1+exp(-2*b_0(j)));
               r = rand(1);
               if r < Pr
                  h(j) = 1;
               else
                  h(j) = -1;
```

```
            end
        end
        % Updating visible neurons
        for t = 1: k
            b_v = (h * w) - t_v;
            for j2 = 1:length(b_v)
                Pr = 1/(1+exp(-2*b_v(j2)));
                r = rand(1);
                if r < Pr
                    v(j2) = 1;
                else
                    v(j2) = -1;
                end
            end
            % Updating hidden neurons
            b_h = (v * w')' - t_h;
            for j3 = 1:M(nrneurons)
                Pr = 1/(1+exp(-2*b_h(j3)));
                r = rand(1);
                if r < Pr
                    h(j3) = 1;
                else
                    h(j3) = -1;
                end
            end
        end
        dt_v = dt_v - eta*(v0-v);
        dt_h = dt_h - eta*(tanh(b_0)-tanh(b_h));
        dw = dw + eta*((tanh(b_0)*v0) - tanh(b_h)*v);
    end
    t_v = t_v + dt_v;
    t_h = t_h + dt_h;
    w = w + dw;
end % Done with training

% Iterating over all x-inputs
outer = 10^3;
inner = 10^2;
Pb = zeros(total_x,1);
T = outer*inner;
% Outer, updating hidden neurons
for i = 1:outer
    idx = randi(total_x);
    v = inputs(idx, :)';
```

```
        b_01 = (w * v) - t_h;
        for j = 1:M(nrneurons)
            Pr = 1/(1+exp(-2*b_01(j)));
            r = rand(1);
            if r < Pr
                h(j) = 1;
            else
                h(j) = -1;
            end
        end
        % Inner, updating visible neurons
        for i2 = 1:inner
            b_v1 = (h * w) - t_v;
            for j3 = 1:length(b_v1)
                Pr = 1/(1+exp(-2*b_v1(j3)));
                r = rand(1);
                if r < Pr
                    v(j3) = 1;
                else
                    v(j3) = -1;
                end
            end
            % Updating hidden neurons
            b_h1 = (w * v) - t_h;
            for j4 = 1:M(nrneurons)
                Pr = 1/(1+exp(-2*b_h1(j4)));
                r = rand(1);
                if r < Pr
                    h(j4) = 1;
                else
                    h(j4) = -1;
                end
            end

            % Check for convergence, when the vectors are the same
            for j5 = 1 : total_x
                x_val = inputs(j5,:);
                if isequal(v', x_val)
                    Pb(j5) = Pb(j5) + 1/T;
                end
            end
        end
    end
% Calculations for the Kullback-Leibler divergence
```

```
    DKL_val = 0;
    for i = 1:total_x
        if (P(i) ~= 0)
            DKL_val = DKL_val + (P(i) * (log(P(i))-log(Pb(i))));
        end
    end
    DKL(Counts,nrneurons) = DKL_val;
end
disp(Counts)
end
%% Plotting the experimental DKL
DKLPlot = zeros(1,4);
for i = 1:4
    DKLPlot(i) = mean(DKL(:,i));
end
figure
plot(M,DKLPlot,'ro')
hold on

% Calculating and plotting the theoretical DKL
M_i = 0:10;
DKL_real = zeros(length(M_i),1);
for i = 1 : length(M_i)
    if M_i(i) < 2^(N-1)-1
        DKL_real(i) = N - (log2(M_i(i)+1)) - (M_i(i)+1)/(2^(log2(M_i(i)+1)));
    else
        DKL_real(i) = 0;
    end
end
plot(M_i, DKL_real, 'k-', 'LineWidth',2)
title('Kullback-Leiber divergence theoretical vs experimental','FontSize',16)
legend('Experimental D_{KL}', 'Theoretical D_{KL}')
xlabel('M')
ylabel('D_{KL}')
hold on
```