Homework 2:Classification challenge

Classify the sequence of digits in "xTest2" as well as you can using a neural-network algorithm. For this task, you may use any package/library to design and train the network. Upload your sequence of classifications as a CSV-file "classifications.csv". If you classify the μ^{th}pattern as the digit 6, then write 6 in the μ^{th} entry of the CSV-file, and in the same way for the other digits. It does not matter whether the sequence is written as a row or a column. Use the upload button at the top of this page to upload your file.

load data

```
clc
clear
xTest2 = loadmnist2();
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(3);
```

```
Preparing MNIST data...
MNIST data preparation complete.
```

```
size(xTrain);
size(xTest2);
% one_digit = xTest2(:,:,:,3);
```

Define Network Architecture

```
layers_1 = [
    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];
```

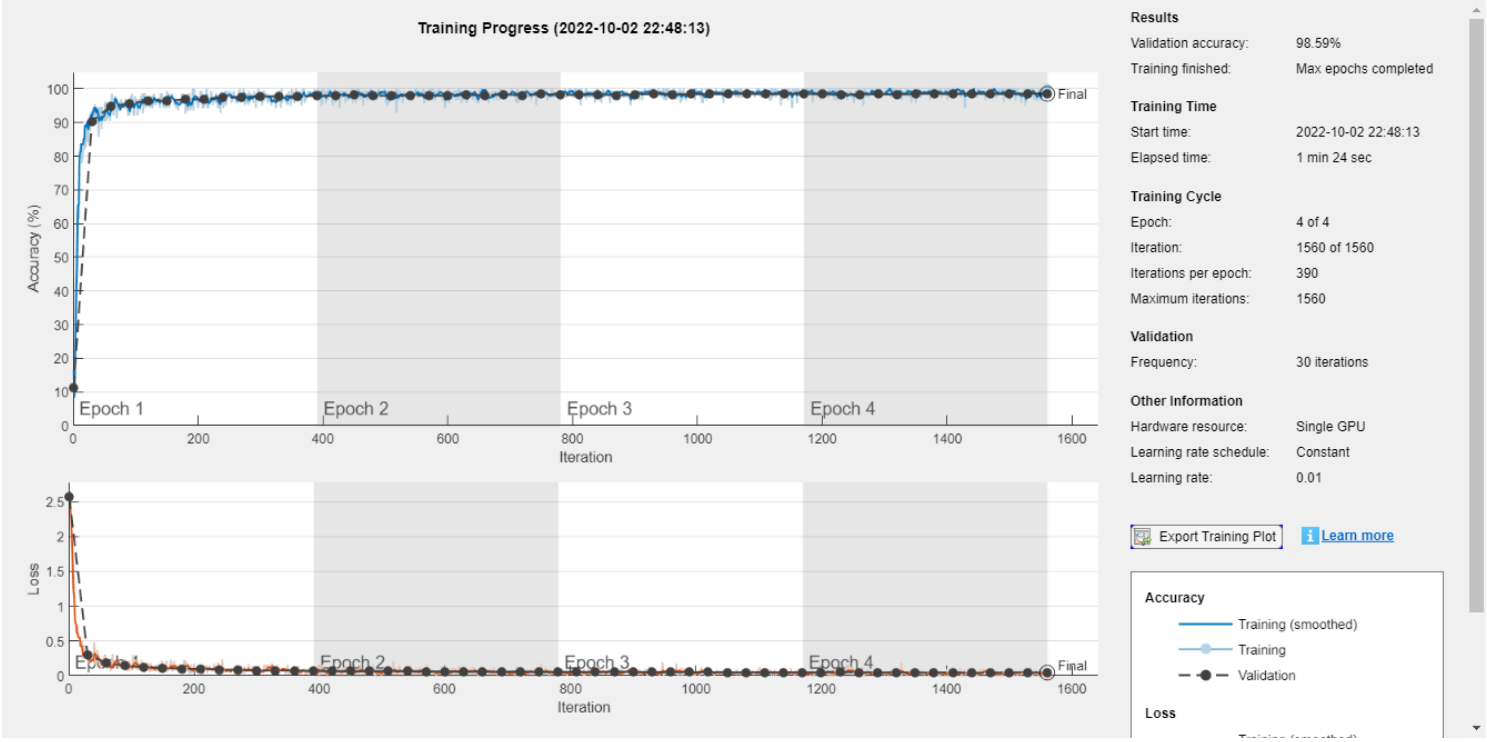
Training Options

```
options_1 = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',{xValid,tValid}, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress')
```

```
options_1 =
  TrainingOptionsSGDM with properties:

                      Momentum: 0.9000
                 InitialLearnRate: 0.0100
                LearnRateSchedule: 'none'
              LearnRateDropFactor: 0.1000
              LearnRateDropPeriod: 10
                 L2Regularization: 1.0000e-04
          GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                        MaxEpochs: 4
                    MiniBatchSize: 128
                          Verbose: 0
                  VerboseFrequency: 50
                    ValidationData: {[28×28×1×10000 uint8]  [10000×1 categorical]}
               ValidationFrequency: 30
                ValidationPatience: Inf
                          Shuffle: 'every-epoch'
                   CheckpointPath: ''
               CheckpointFrequency: 1
            CheckpointFrequencyUnit: 'epoch'
               ExecutionEnvironment: 'auto'
                         WorkerLoad: []
                          OutputFcn: []
                              Plots: 'training-progress'
                     SequenceLength: 'longest'
               SequencePaddingValue: 0
            SequencePaddingDirection: 'right'
               DispatchInBackground: 0
             ResetInputNormalization: 1
    BatchNormalizationStatistics: 'population'
                  OutputNetwork: 'last-iteration'
```

Train Network

```
net_1 = trainNetwork(xTrain,tTrain,layers_1,options_1);
```



Classify Validation Images and Compute Accuracy

```
tValid_pred_1 = classify(net_1, xValid);
accuracy_Val_1 = sum(tValid_pred_1 == tValid)/numel(tValid);

tTest_pred_1 = classify(net_1, xTest);
accuracy_Test_1 = sum(tTest_pred_1 == tTest)/numel(tTest);

tTest2_pred_1 = classify(net_1, xTest2);
disp(accuracy_Val_1)
```

```
    0.9859
```

```
disp(accuracy_Test_1)
```

```
    0.9863
```

```
writematrix(tTest2_pred_1,"classifications.csv")
```

Homework 2: One-layer perceptron

$\underline{x}^{(\mu)} = [x_1^{(\mu)}, x_2^{(\mu)}]^T$

$t^{\mu} = \pm 1$

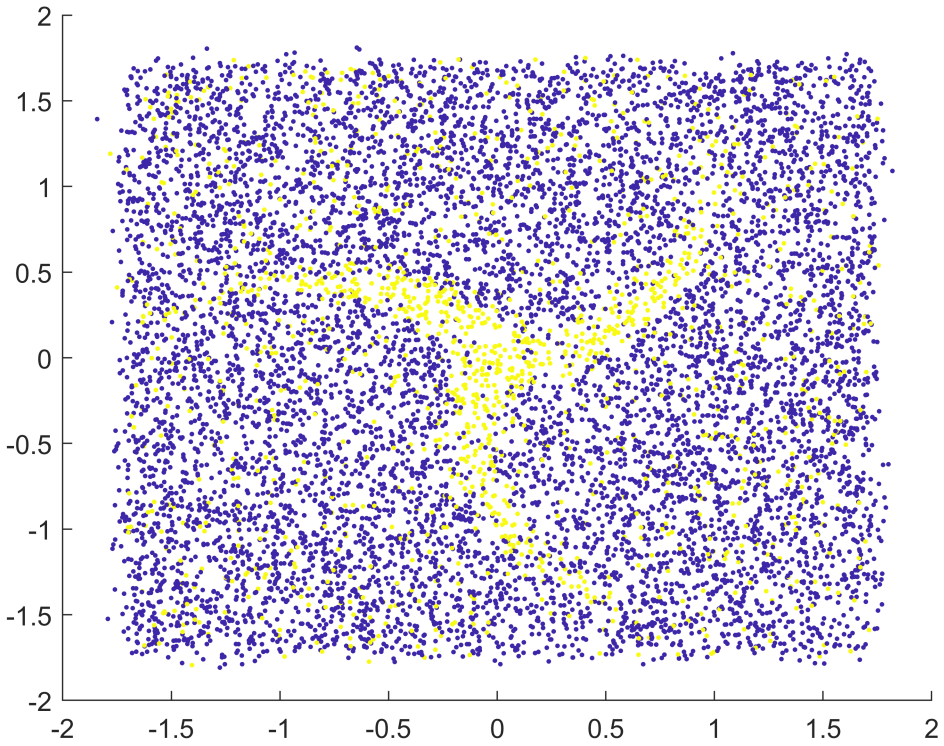
Start by normalizing the data by centering each of the two input components and normalizing their respective variances to 1.

Your task is to train a perceptron with one fully-connected hidden layer, two input terminals and one output unit. Use the following network layout:

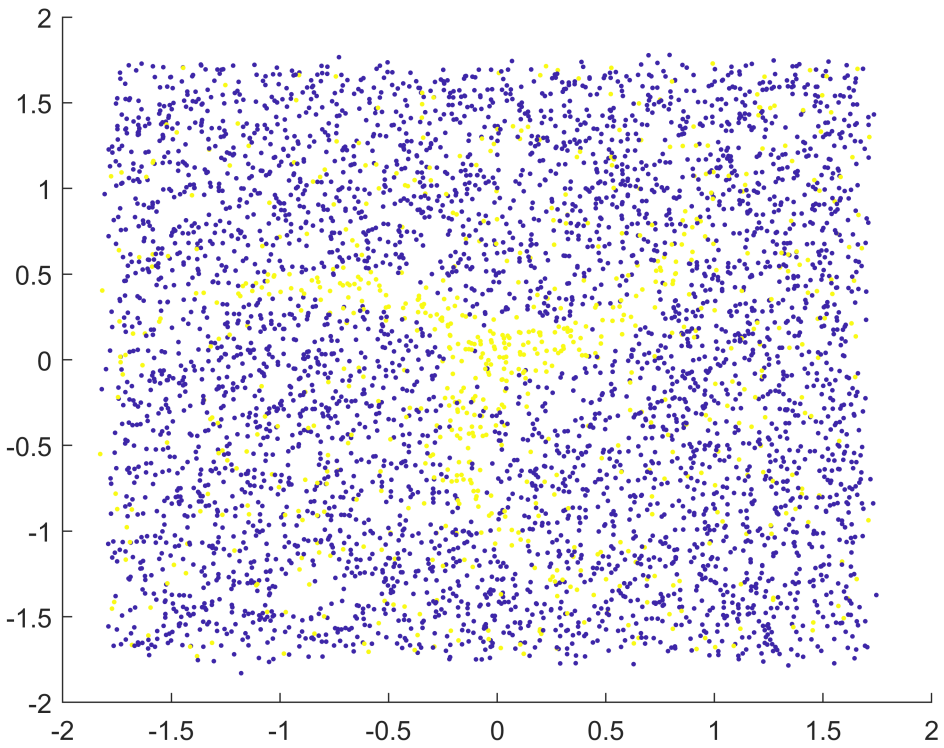
```
clc
clear
training_set = readmatrix("training_set.csv");
validation_set = readmatrix("validation_set.csv");
eta = 0.006;
N = 2;
epoch_n = 2000;
error_criteria = 0.12;
```

Preprocessing

```
train_input_raw = training_set(:,1:2);
train_input = normalize(train_input_raw)';      % [2x10000]
target = training_set(:,3);                     % [10000x1]
figure
scatter(train_input(1,:),train_input(2,:),[],target,'.');
```




```
p = size(train_input,2);

figure
validation_input_raw = validation_set(:,1:2);
validation_input = normalize(validation_input_raw)';
% validation_input = validation_set(:,1:2)';      % [2x5000]
validation_target = validation_set(:,3);        % [5000x1]
scatter(validation_input(1,:),validation_input(2,:),[],validation_target,'.')
```




```
p_val = size(validation_input,2);
```

Initialization

Different number of hidden neurons in the hidden layer.

```
M = 10;  % M goes to 100???
w = normrnd(0, 1/M, [M,2]);     % [Mx2] w_jk
theta = zeros([M,1]);           % [Mx1] theta_j
W = normrnd(0, 1/M, [1, M]);    % [1xM] W_ij i=1
Theta = 0;
V = zeros([M,1]);               % [Mx1] V_j
Out = 0;
H_train = [];
H_val = [];
```

Set energy function

$$H = \frac{1}{2}\sum_{\mu i}\left(t_i^{(\mu)} - O_i^{(\mu)}\right)^2 . \qquad (6.4)$$

```
H_train(end+1) = energy(W, w, Theta, theta, train_input, target);
H_val(end+1) = energy(W, w, Theta, theta, validation_input, validation_target);
```

Training

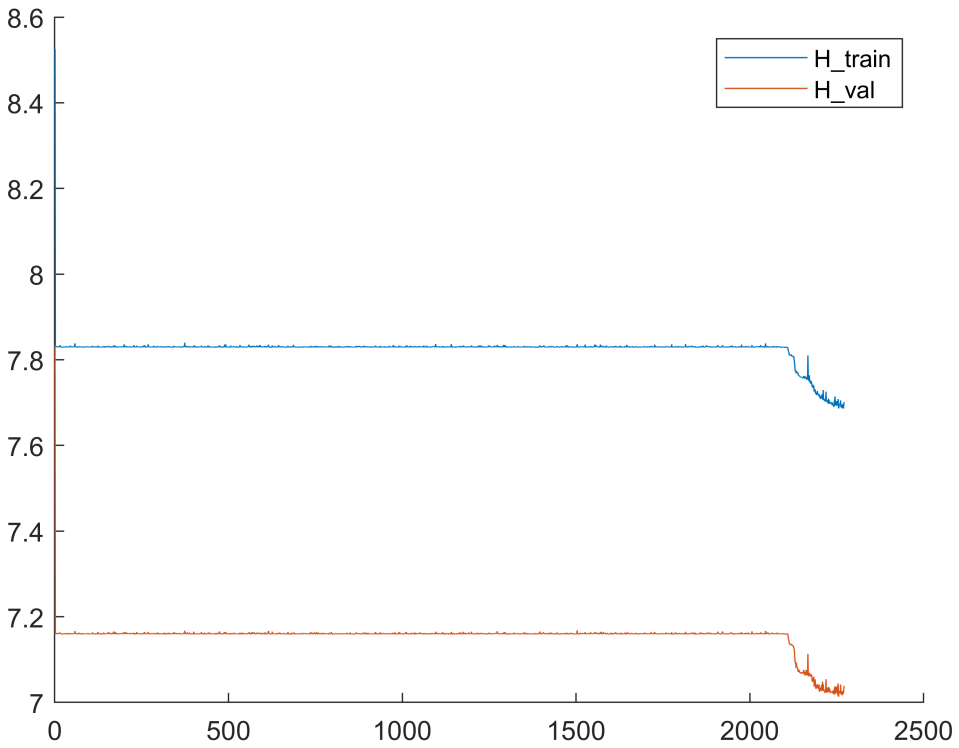
```
for ep=1:epoch_n
    for nu = 1 : p
        mu = randi([1,p]);
        x_mu = train_input(:,mu); % randomly pick an input from training set
        t_mu = target(mu);

        % propagate forward
        b_mu = w*x_mu-theta;    % [3x1]
        V = tanh(b_mu);         % activation function g = tanh()
        B_mu = W*V-Theta;  % scalar
        Out = tanh(B_mu);       % scalar
        % end of propagate forward

        % back propagation
        %% Calculate Delta
%         Delta = (target(mu) - Out) * eval(subs(diff(g), B_mu));
        Delta = (target(mu) - Out) * (1-tanh(B_mu)^2);

        %% Calculate delta
%         delta = Delta * W'.* eval(subs(diff(g), b_mu));
%         delta = Delta * W'.* (1-tanh(b_mu).^2);
        delta = Delta * W'.* (1-tanh(b_mu).^2);

        %% update W and w
        delta_w = eta * delta * x_mu';
        delta_W = eta * Delta * V';

        W = W + delta_W;
        w = w + delta_w;

        %% update Theta
        delta_Theta = -eta * Delta;
        Theta = Theta + delta_Theta;

        %% update theta
        delta_theta = -eta*delta;
        theta = theta + delta_theta;
        % end of back propagation
    end
    W;
    w;
    H_train(end+1) = energy(W, w, Theta, theta, train_input, target);
    H_val(end+1) = energy(W, w, Theta, theta, validation_input, validation_target);

%     plot(1:length(H_train),log(H_train))
%     plot(1:length(H_val),log(H_val))
%     legend('H\_train','H\_val')
```

```
    b = w*validation_input-theta;
    V = tanh(b);
    B = W*V-Theta;
%     C = 1/(2*p_val)*sum(abs(validation_target'-sign(B)));
    tmp = 0;
    for mu = 1:p_val
        m = randi(p_val);
%         m = mu;
        x_mu = validation_input(:,m);
        b_mu = w * x_mu - theta;    % [Mx1]
        V = tanh(b_mu);         % activation function g = tanh()
        B_mu = W * V - Theta;  % scalar
        Out = tanh(B_mu);       % scalar
        tmp = tmp + abs(validation_target(m)-sign(Out));
    end
    C = 1/(2*p_val)*tmp;
    if C < error_criteria
        disp('Classification error criteria is met');
        disp(['Epoch: ',num2str(ep),' C: ',num2str(C)])
        break
    else
        disp(['Epoch: ',num2str(ep),' C: ',num2str(C)])
    end
end
```

```
Classification error criteria is met
Epoch: 270 C: 0.1184
```

Plot H

```
figure
hold on
plot(1:length(H_train),log(H_train))
plot(1:length(H_val),log(H_val))
legend('H\_train','H\_val')
hold off
```




```
w
```

```
w = 10×2
   -1.5759    2.0822
   -0.3253   -0.1967
    3.5666    2.1493
   -4.0089    5.1979
    4.1902    1.0640
    0.6088    0.2444
   -0.4945   -4.6993
    0.3614    0.0382
    0.6054    1.6781
    2.7554   -1.4534
```

```
W
```

```
W = 1×10
    1.1119   -0.2276   -2.0984   -1.9447    1.6450    0.5558   -2.5909    0.2866 ···
```

```
theta
```

```
theta = 10×1
   -0.3132
   -0.0148
   -0.0026
    0.0715
   -1.4094
   -0.1113
    0.5144
   -0.0027
   -1.0124
```

```
    1.7783
```

```
Theta
```

```
Theta = 2.8069
```

```
csvwrite("w1.csv",w)
csvwrite("w2.csv",W')
csvwrite("t1.csv",theta)
csvwrite("t2.csv", Theta)
```

```
function Out = fd_prop(W, w, Theta, theta, x)
    b_mu = w * x - theta;    % [Mx1]
    V = tanh(b_mu);          % activation function g = tanh()
    B_mu = W * V - Theta;  % scalar
    Out = tanh(B_mu);        % scalar
end

function H = energy(W, w, Theta, theta, input_set, target_set)
    H = 0;
    for mu = 1:size(input_set, 2)
        Out = fd_prop(W,w,Theta,theta,input_set(:,mu));
        H = H + (target_set(mu)-Out)^2;
    end
    H = H * 0.5;
end
```

Homework 2: Restricted Boltzmann machine

The goal of this exercise is to train a restricted Boltzmann machine to learn the XOR data set.

In the XOR data set, the three-bit patterns shown in the Figure to the right are assigned probability $\frac{1}{4}$, all other patterns are assigned probability zero. Here □ corresponds to -1, and ■ to +1. See Figure 4.5 in the lecture notes.

Train a restricted Boltzmann machine with three visible and M=1,2,4,8 hidden neurons (all +/-1 neurons) using the CD-k algorithm. Experiment with different values for k and for the learning rate η. Make sure that you use the correct algorithm for +/-1 neurons (Algorithm 3 in the course book).

Compute the Kullback-Leibler divergence as a function of the number of hidden neurons: iterate the dynamics of the restricted Boltzmann machine after training, and count the frequencies at which the different patterns occur. Determine for how long you must run the dynamics to get a precise estimate of the probabilities. Plot the Kullback-Leibler divergence vs. M. Also plot the theory [Equation (4.40)], compare, and discuss your results.

Parameters

```
clc
clear
x1 = [-1;-1;-1];
x2 = [1;-1;1];
x3 = [-1;1;1];
x4 = [1;1;-1];
x = [x1,x2,x3,x4];
x5 = [-1;-1;1];
x6 = [1;-1;-1];
x7 = [-1;1;-1];
x8 =[1;1;1];
x_all = [x1,x2,x3,x4,x5,x6,x7,x8];
P_data = 1/4;
counter = 5;
trials = 1000;
minibatch_n = 20;
k = 500;
Nout = 3000;
Nin = 2000;
NN = Nout*Nin;
% M = 4;
M_list = [1,2,3,4,8];
N = 3;
% V = zeros([N,1]);
% H = zeros([M,1]);
% w = normrnd(0, 1/sqrt(N), [M,N]);
% theta_v = zeros([N,1]);
% theta_h = zeros([M,1]);
eta = 0.003;%3
```

```
D_KL_list=[];
```

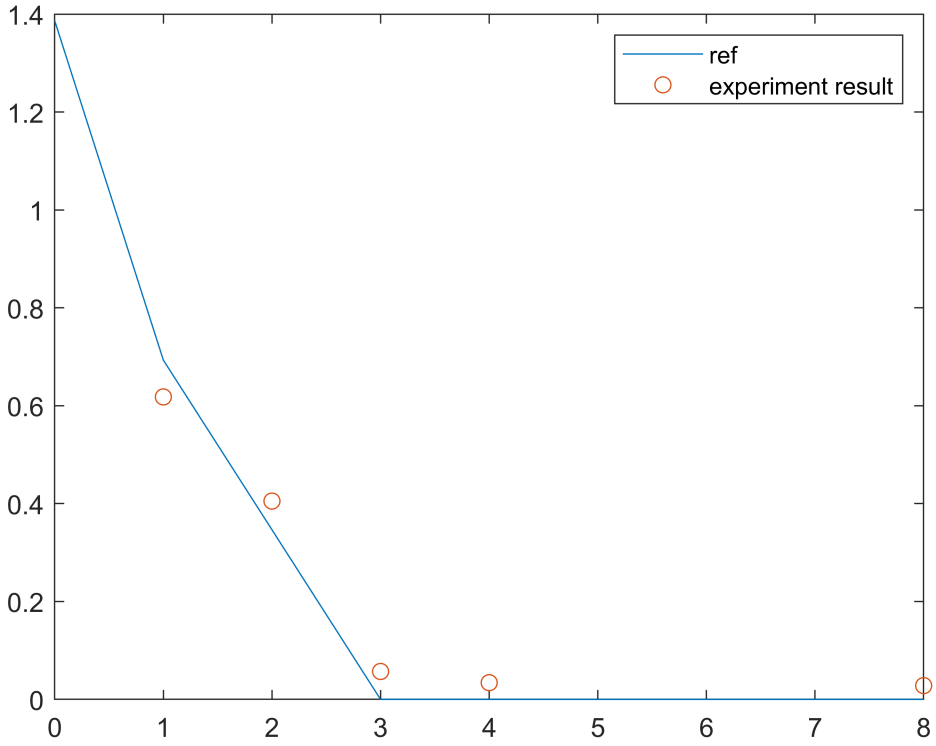
```
for iter = 1:length(M_list)  % loop through M=1,2,3,4,8
    M = M_list(iter);
    D_KL_minimal = 5;
    for count = 1:counter % convergence over D_KL
        V = zeros([N,1]);
        H = zeros([M,1]);
        w = normrnd(0, 1/sqrt(N), [M,N]);
        theta_v = zeros([N,1]);
        theta_h = zeros([M,1]);
        for trial = 1:trials % convergence over w theta_v theta_h
            delta_w = zeros([M,N]);
            delta_theta_v = zeros([N,1]);
            delta_theta_h = zeros([M,1]);
        %     input_batch = [x(:,randi(4)),x(:,randi(4)),x(:,randi(4))]; % sample 3 patterns fr
            for p = 1:minibatch_n % feed all the patterns in the minibatch
                input_pattern = x_all(:,randi(4)); % [3x1]
                V_0 = input_pattern;
                V = input_pattern;
                b_h_0 = w*V_0 - theta_h;
                % update Hidden neurons
                b_h = w*V - theta_h; % [Mx1]
                for idx = 1:M
                    r = rand(1);
                    if r < P_Boltz(b_h(idx))
                        H(idx) = 1;
                    else
                        H(idx) = -1;
                    end
                end

                % loop over CD-k
                for t = 1:k
                    % update all visible neurons
                    b_v = w' * H - theta_v; % [3x1]
                    for idx = 1:N
                        r = rand(1);
                        if r < P_Boltz(b_v(idx))
                            V(idx) = 1;
                        else
                            V(idx) = -1;
                        end
                    end
                    % update all hidden neurons
                    b_h = w*V - theta_h; % [Mx1]
                    for idx = 1:M
                        r = rand(1);
```

```
                    if r < P_Boltz(b_h(idx))
                        H(idx) = 1;
                    else
                        H(idx) = -1;
                    end
                end
            end % end of loop over k times

            % compute weight and threshold increments delta_w_mn
            for m = 1:M %1->M
%                 bm_h_0 = w(m,:)*V_0 - theta_h(m);
                bm_h_k = w(m,:)*V - theta_h(m);
                for n = 1:length(V) % 1->3
                    delta_w(m,n) =  delta_w(m,n) + eta * (tanh(b_h_0(m))*V_0(n)-tanh(bm_h_
%                     delta_theta_v(n) = delta_theta_v(n) - eta * (V_0(n)-V(n));
                end
                delta_theta_h(m) = delta_theta_h(m) - eta * (tanh(b_h_0(m)) - tanh(bm_h_k)
            end % end of calculate delta's
            delta_theta_v = delta_theta_v - eta * (V_0-V);
        end % end of minibatches

        % update weights and threshold
        w = w + delta_w;
        theta_v = theta_v + delta_theta_v;
        theta_h = theta_h + delta_theta_h;
    end % end of trials, trainings

    % Kullback-Leibler divergence.
    P_B = zeros([1,8]);
    % start the outer loop
    for l = 1:Nout
        p_selected = x_all(:,randi(8));
        V = p_selected;

        % update Hidden neurons
        b_h = w*V - theta_h; % [Mx1]
        for idx = 1:M
            r = rand(1);
            if r < P_Boltz(b_h(idx))
                H(idx) = 1;
            else
                H(idx) = -1;
            end
        end

        % start the inner loop
        for t = 1:Nin
            % update all visible neurons
            b_v = w' * H - theta_v; % [3x1]
            for idx = 1:N
```

```
                    r = rand(1);
                    if r < P_Boltz(b_v(idx))
                        V(idx) = 1;
                    else
                        V(idx) = -1;
                    end
                end
                % update all hidden neurons
                b_h = w*V - theta_h; % [Mx1]
                for idx = 1:M
                    r = rand(1);
                    if r < P_Boltz(b_h(idx))
                        H(idx) = 1;
                    else
                        H(idx) = -1;
                    end
                end

                % check which pattern of dataset is V
                for idx = 1:8
                    if isequal(x_all(:,idx), V)
                        P_B(idx) = P_B(idx) + 1;
                    end
                end
            end % end of inner loop
        end % end of outer loop
        P_B = P_B/NN;
        D_KL = 0;
        for idx = 1:4
            D_KL = D_KL + P_data * log(P_data/P_B(idx));
        end
        if D_KL < D_KL_minimal
            D_KL_minimal = D_KL;
        end
    end
    D_KL_list(end+1) = D_KL_minimal;
end
```

```
D_KL_Blist = [];
M_list_rf =[0,1,2,3,4,8];
for iter = 1:length(M_list_rf)
    M = M_list_rf(iter);
    if M < 2^(N-1)-1
        D_KL_bound = log(2)*(N-fix(log2(M+1))-(M+1)/(2^fix((log2(M+1)))));
%         D_KL_bound = N-fix(log2(M+1))-(M+1)/(2^fix((log2(M+1))));
    else
        D_KL_bound = 0;
    end
    D_KL_Blist(end+1) = D_KL_bound;
```

```
end
figure
plot(M_list_rf,D_KL_Blist)
hold on
scatter(M_list, D_KL_list)
legend("ref","experiment result")
```




```
function p = P_Boltz(b)
    p = 1/(1+exp(-2*b));
end
```

Homework 2 Restricted Boltzmann machine

In my training program, the CD-k will iterate 1000 times with $k = 500$ and mini-batches with 20 randomly selected patterns. $\eta = 0.003$. The training process will repeat for 5 times to find the minimal D_{KL}.

In calculating Kullback-Leibler divergence, 3000 patterns are fed into the trained neural network then update the visible and hidden neurons back and forth for 2000 rounds. Accumulate the times that each pattern shows up, then this will be used to calculate the distribution that the Boltzmann machine approximates, P_B.

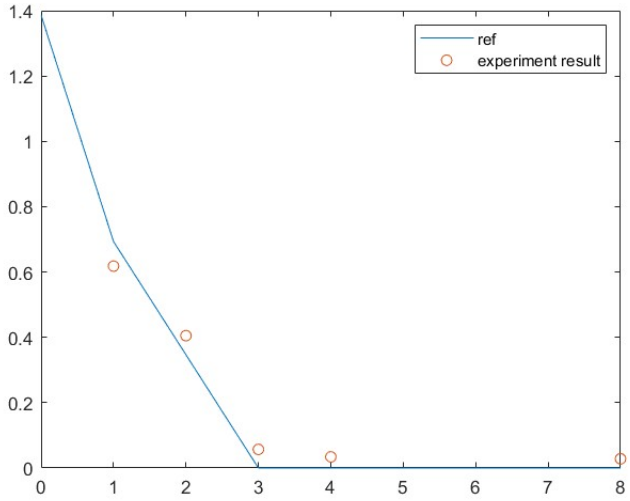


Figure 1: Numerical results

The figure above illustrates how the Kullback-Leibler divergence changes according to different numbers of hidden neurons in a restricted Boltzmann machine. The experiment result is roughly above the upper boundary. It could be because the CD-k algorithm is not an optimal method. And since calculating D_{KL} is a stochastic process, the value can be more precise under a large number of tests.

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

Small k or large η will result in the model not converging. The corresponding D_{KL} will so that not stay below the boundary.