Restricted Boltzmann machine

FFR135 Artificial Neural Networks

Axel Johansson
axejoh@student.chalmers.se

October 6, 2022

Solution proposal

The task is to train a restricted Boltzman machine (RBM) to learn four patterns in the XOR data set. The four 3-bit patterns are

$$\text{Data} = \begin{bmatrix} -1 & -1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \tag{1}$$

and should each be returned with probability 25%.

The RBM does not assign weights with Hebb's rule like in the Hopefield network. Instead they are assigned by iterating back and forth through the network updating weights and thresholds until the Kullback-Leibler divergence is minimized i.e. the RBM distribution approximates the data distribution. The update rule is a special case of the Markov-chain Monte-Carlo algorithm for Hopefield with energy function. The learning rule is derived using gradient ascent on the log-likelihood. The network is trained using a contrastive-divergence algorithm.

The RBM was trained for $M = [1,2,4,8]$ hidden neurons and the KL divergence are shown in tabular 1 and figure 1. The divergence is nonzero for hidden neurons fewer than 4. Therefore only one or two hidden neurons is not sufficient for the RBM to store the four patterns perfectly. This finding is consistent with the book which states in [Equation (4.40)] that in general $M = 2^{N-1}$ hidden neurons are sufficient to reach a small divergence.

Table 1: Kullback-Leibler divergence over the number of neurons M.

M	KL divergence
1	0,70
2	0,46
4	0.02
8	0.02

The KL divergence increased for 8 neurons when trained with the same number of CD-k iterations. This was inceased to achive a lower value. The Settings used is presented in tabular 2.

Table 2: The settings used to train the network is presented in the tabular.

Name	Value
eta	0.002
nTrials	1000
nEpochs	200
nCdk	3000

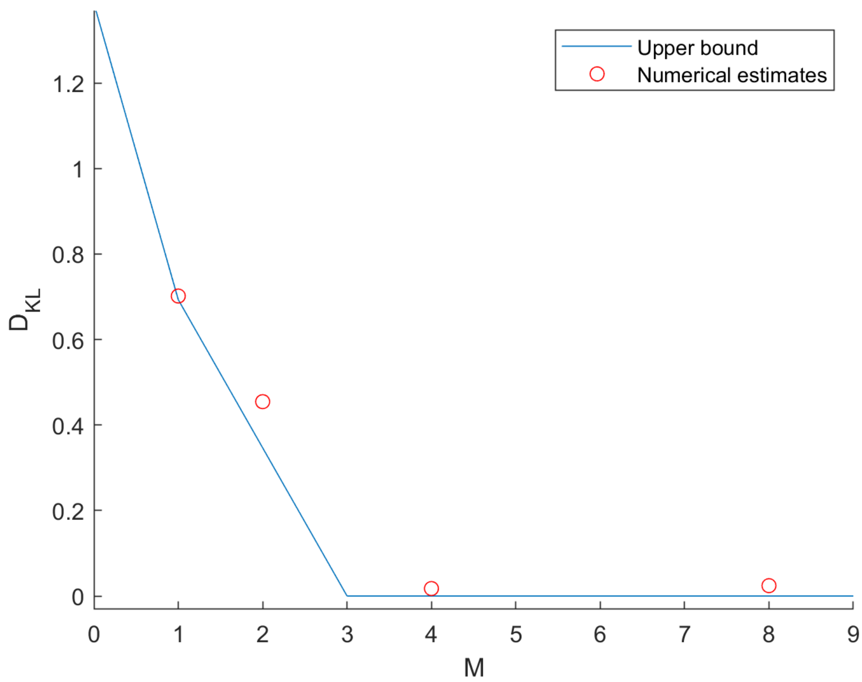

Figure 1: *The figure shows the KL divergence for different values of M in red against an upper bound in blue. As can be seen the convergence is under the upper bound. For* $M > 2^{N-1}$ the KL divergence goes to 0 in the figure which is expected from Equation 4.40 in the book.

1 MATLAB-kod

```
%% Boltzmann machine
clc, clear all
% Settings%%%%%%%%%%%%%%%%%%%%%%%
%M = 4;% = [1,2,4,8];
eta = 0.002;
nTrials = 1000; % 1000 run with 2000
nEpochs = 200; % 20
nCdk = 3000; % 20
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initializing
input = [-1 -1 -1; 1 -1 1; -1 1 1; 1 1 -1];
data = dec2bin(0:7)-'0';
data(data == 0) = -1;
pData = [1/4 0 0 1/4 0 1/4 1/4 0];

ProbDistr = zeros(4, length(data));
Miteration = 0;

for M = [1 2 4 8]
PB = zeros(1, length(data));
w = normrnd(0, 1,[M, size(input,2)]);
w(1:1+size(w,1):end) = 0;
thetaV = zeros(1,size(input,2));
thetaH = zeros(1,M);
Miteration = Miteration +1;

for i = 1:nTrials
    dw = zeros(M, size(input,2));
    dthetaV = zeros(1, size(input,2));
    dthetaH = zeros(1,M);

    for j = 1:nEpochs
        v0 = input(randi([1 size(input, 1)]),:);
        bh0 = (w * v0') - thetaH';
        P = 1./(1 + exp(-2*bh0));
        h = UpdateNeuron(M, P);

        for cdk = 1:nCdk
            bv = (h'*w) - thetaV;
            P = 1./(1 + exp(-2*bv));
            v = UpdateNeuron(size(input,2), P)';

            bh = (w * v') - thetaH';
            P = 1./(1 + exp(-2*bh));
            h = UpdateNeuron(M, P);
        end
        dw = dw + eta * (tanh(bh0)*v0-tanh(bh)*v);
        dthetaV =  dthetaV  -eta * (v0-v);
        dthetaH = dthetaH -eta * (tanh(bh0')-tanh(bh'));

    end
    w = w + dw;
    thetaV = thetaV + dthetaV;
    thetaH = thetaH + dthetaH;
end

Nout = 3000;
Nin = 2000;

for outer = 1:Nout
```

```
    r = randi([1 length(data)]);
    v = data(r,:);
    bh = (w * v') - thetaH;
    P = 1./(1 + exp(-2*bh));
    h = UpdateNeuron(M, P);

    for inner = 1:Nin
        bv = (h'*w) - thetaV;
        P = 1./(1 + exp(-2*bv));
        v = UpdateNeuron(size(input,2), P).';

        bh = (w * v') - thetaH';
        P = 1./(1 + exp(-2*bh));
        h = UpdateNeuron(M, P);
         for j = 1:length(data)
                if isequal(v, data(j,:))
                    PB(j) = PB(j) + 1;
                end
         end
    end
    outer
end
ProbDistr(Miteration,:) = PB/(Nin*Nout);
end

DKL = zeros(4,8);
for M = 1:4
    for r = 1:8
        if ProbDistr(M,r) == 0 && pData(r) == 0
            DKL(M,r) = 0;
        elseif pData(r) == 0
            DKL(M,r) = pData(r).*(0-log(ProbDistr(M,r)));
        elseif ProbDistr(M,r) == 0
            DKL(M,r) =  pData(r).*(log(pData(r)));
        else
            DKL(M,r) =  pData(r).*(log(pData(r))-log(ProbDistr(M,r)));
        end
    end
end

hold on
x = 0:0.02:9;
KL = log(2)*(size(input,2) - (fix(log2(x+1)))-(x+1)./(2.^fix((log2(x+1)))));
KL(KL<0) = 0;
plot(x,KL)
DKL = abs(sum(DKL,2))
plot([1 2 4 8],DKL,'O','color', 'red')
xlabel('M')
ylabel('D_{KL}')
legend({'Upper bound','Numerical estimates'},'Location','northeast')
hold off

function h = UpdateNeuron(M, P)
    h = zeros(M,1);
    for k = 1:M
        if rand < P(k)
            h(k) = 1;
        else
            h(k) = -1;
        end
    end
end
```