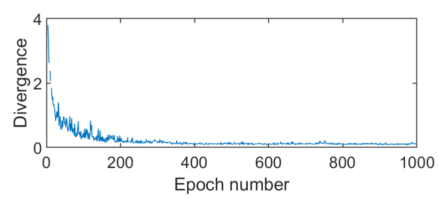
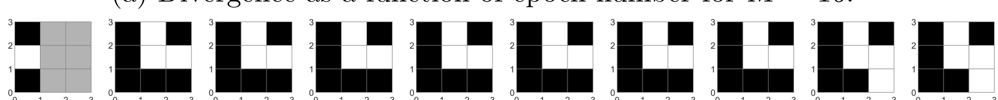
Restricted Boltzmann machine

David Tonderski (davton)

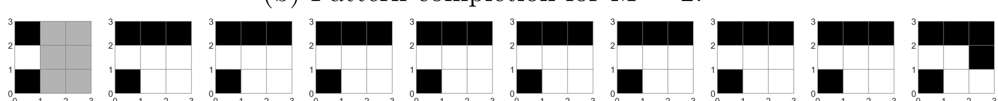
The divergence was calculated as follows: in each epoch, each *possible* pattern was fed to the Boltzmann machine. The dynamics were then run for 100 iterations, and the frequencies at which the different *possible* patterns occured were counted. This was then used as an approximation for the model distribution P_B for each epoch, and the Kullback-Leibler divergence was calculated using $d = \sum_\mu P_D(\mu) \log(P_D(\mu)/P_B(i_\mu))$, where $P_D(\mu) = \frac{1}{14}$ for all data set patterns μ, and i_μ is the index of the data set pattern μ in the set of all possible patterns. If $P_B(\mu) = 0$, I set $d = \infty$. For $M = 2, 4, 8$, the divergence was infinity for (almost) all epochs, so the plots are not shown. For $M = 16$, the divergence is shown in figure 1a. The first 10 produced patterns after feeding the first column are shown in figures 1b, 1c, 1d, 1e.



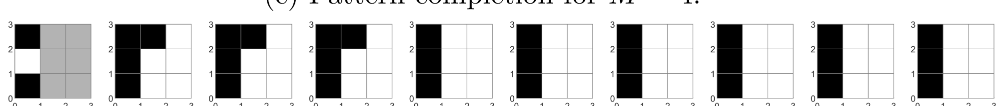
(a) Divergence as a function of epoch number for $M = 16$.



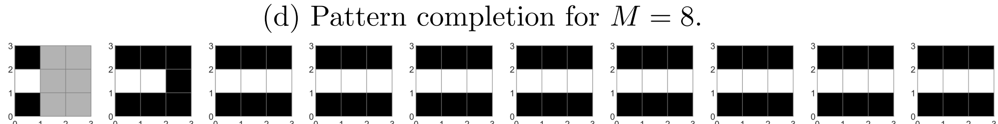
(b) Pattern completion for $M = 2$.



(c) Pattern completion for $M = 4$.



(d) Pattern completion for $M = 8$.



(e) Pattern completion for $M = 16$.

Figure 1: Training information for the two networks.

As expected, the model works well for $M = 16$, and not at all for $M = 2, 4$. Interestingly, the model converges to a data set pattern for $M = 8$, despite the divergence being ∞, but it is not the correct one. This probably means that the model has learned a few of the data set patterns, but not all of them.

Restricted Boltzmann machine

Table of Contents

Initialization

```
clear; close all
```

Patterns

```
patterns = [[-1;-1;-1;-1;-1;-1;-1;-1;-1], ...
            [ 1;-1;-1; 1;-1;-1; 1;-1;-1], ...
            [-1; 1;-1;-1; 1;-1;-1; 1;-1], ...
            [-1;-1; 1;-1;-1; 1;-1;-1; 1], ...
            [ 1; 1;-1; 1; 1;-1; 1; 1;-1], ...
            [-1; 1; 1;-1; 1; 1;-1; 1; 1], ...
            [ 1;-1; 1; 1;-1; 1; 1;-1; 1], ...
            [ 1; 1; 1; 1; 1; 1; 1; 1; 1], ...
            [ 1; 1; 1;-1;-1;-1;-1;-1;-1], ...
            [-1;-1;-1; 1; 1; 1;-1;-1;-1], ...
            [-1;-1;-1;-1;-1;-1; 1; 1; 1], ...
            [ 1; 1; 1; 1; 1; 1;-1;-1;-1], ...
            [-1;-1;-1; 1; 1; 1; 1; 1; 1], ...
            [ 1; 1; 1;-1;-1;-1; 1; 1; 1]];

possiblePatterns = zeros(9,512);

for i = 1:512
    possiblePatterns(:,i) = DecimalToState(i);
end

patternsDecimal = zeros(1, 14);
for i = 1:14
    patternsDecimal(i) = StateToDecimal(patterns(:,i));
end
```

Parameters

```
MArray                      =   [2 4 8 16];            % Hidden
 Neurons
N                           =   9;              % Input
max_epochs                  =   1000;
```

```
p                           =   14;
beta                        =   1;
iterationLength             =   100;
eta                         =   0.01;
verbose                     =   1;
frequencySum                =   1400;
PData                       =   1/14;
repeatsPerPattern           =   1;
divergenceIterationLength   =   100;
```

Training and plotting

```
iFigure = 1;
for M = MArray
    weightMatrix                =   -1+2*rand(M,N);
    thetaV                      =   -1+2*rand(1,N);
    thetaH                      =   -1+2*rand(M,1);
    divergenceArray             =   zeros(1, max_epochs);
    for iEpoch = 1:max_epochs
        frequency = zeros(512, 1);
        if(verbose && mod(iEpoch, max_epochs/10) == 0)
            fprintf('M is %d, Epoch is %d, %d percent done.\n', M,
 iEpoch, round(iEpoch/max_epochs*100));
        end
        deltaWeight = zeros([size(weightMatrix), 14]);
        deltaThetaV = zeros([size(thetaV), 14]);
        deltaThetaH = zeros([size(thetaH), 14]);
        for mu = 1:14
            pattern = patterns(:,mu);
            correctDecimal = StateToDecimal(pattern);
            v = pattern;
            for t = 1:iterationLength
                v = RunIteration(v, weightMatrix, beta, thetaV,
 thetaH);
            end
            deltaWeight(:,:,mu)  =  eta*(tanh(weightMatrix*pattern -
 thetaH)*pattern' - tanh(weightMatrix*v - thetaH)*v');
            deltaThetaV (:,:,mu) = -eta*(pattern - v);
            deltaThetaH (:,:,mu) = -eta*(tanh(weightMatrix*pattern -
 thetaH) - tanh(weightMatrix*v - thetaH));
        end
        weightMatrix = weightMatrix + sum(deltaWeight,3);
        thetaV       = thetaV +       sum(deltaThetaV,3);
        thetaH       = thetaH +       sum(deltaThetaH,3);

        divergenceArray(iEpoch) =
 GetKullbackLeiblerDivergence(weightMatrix, ...
            repeatsPerPattern, divergenceIterationLength,
 possiblePatterns, ...
            patternsDecimal, beta, thetaV, thetaH);
    end

    figure(iFigure)
```

```
    clf
    plot(divergenceArray);
    xlabel('Epoch number')
    ylabel('Divergence')
    iFigure = iFigure + 1;

    figure(iFigure)
    clf
    middleAndRightColumnIndices = [2,3,5,6,8,9];
    v = patterns(:,14);
    v(middleAndRightColumnIndices) = 0;
    for iteration = 1:10
        subplot(1,10,iteration);
        PlotPattern(v);
        v = RunIteration(v, weightMatrix, beta, thetaV, thetaH);
    end
    iFigure = iFigure + 1;
end
```

Functions

```
function divergence = GetKullbackLeiblerDivergence(weightMatrix, ...
    repeatsPerPattern, iterationLength, possiblePatterns, ...
    patternsDecimal, beta, thetaV, thetaH)

    frequency = zeros(512, 1);
    for t = 1:512*repeatsPerPattern
        v = possiblePatterns(:,ceil(t/repeatsPerPattern));

        for i = 1:iterationLength
            v = RunIteration(v, weightMatrix, beta, thetaV, thetaH);
            decimalRepresentation = StateToDecimal(v);
            frequency(decimalRepresentation) =
 frequency(decimalRepresentation) + 1;
        end
    end

    frequencySum = 512*repeatsPerPattern*iterationLength;
    PB = frequency/frequencySum;
    PData = 1/14;
    divergence = 0;
    for mu = 1:14
        patternIndex = patternsDecimal(mu);
        divergence = divergence+PData*log(PData/PB(patternIndex));
    end
end

function v = RunIteration(v, weightMatrix, beta, thetaV, thetaH)
    b_h = (weightMatrix*v) - thetaH;
    h   = GetNextNeuronState(b_h,beta);
    b_v = (h*weightMatrix) - thetaV;
    v   = GetNextNeuronState(b_v, beta);
end
```

```
function number = StateToDecimal(state)
    state(state==-1) = 0;
    number = 1;
    for j = 1:9
        number = number + state(j)*2^(9-j);
    end
end

function state = DecimalToState(number)
    number = number-1;
    state = zeros(9, 1);
    for j = 1:9
        state(10-j) = mod(number,2);
        number = floor(number/2);
    end
    state(state == 0) = -1;
end

function s = GetNextNeuronState(b, beta)
    b = b';
    p = 1./(1+exp(-2*b*beta));
    s = zeros(size(b));
    for i = 1:length(p)
        r = rand;
        if r > p(i)
            s(i) = 1;
        else
            s(i) = -1;
        end
    end
end

function PlotPattern(pattern)
    image = reshape(pattern, 3, 3)';
    for x = 1:3
        for y = 1:3
            if image(y,x) == 1
                rectangle('Position', [x-1, 3-y, 1, 1], 'FaceColor',[0
 0 0]);
            elseif image(y,x) == 0
                rectangle('Position', [x-1, 3-y, 1, 1], 'FaceColor',
[0.7 0.7 0.7]);
            elseif image(y,x) == -1
                rectangle('Position', [x-1, 3-y, 1, 1], 'FaceColor',[1
 1 1]);
            end
            rectangle('Position', [x-1, 3-y, 1, 1], 'EdgeColor', [0.5
 0.5 0.5]);
        end
    end
    xlim([0 3])
    ylim([0 3])
```

```
end
```

Published with MATLAB® R2020a