Problem1 Assignment1 Isac Nordin

isacn

September 2021

1 code

```
%main code
p = [12,24,48,70,100,120];
N = 120;
trial = 10^5;

p_error = [];
for i = 1:length(p)
    p_error = [p_error; Perror(p(i),N,trial)];
end
disp(round(p_error,4))

%calculates Perror for 1 timestep
function pErr=Perror(nrOfPatterns,nrOfBits,nrOfTrials)

    pErr = 0;
    for iTrial = 1:nrOfTrials
        patterns = sign(2*rand(nrOfPatterns,nrOfBits)-1);
        weightMatrix = GetWeights(patterns);
        %update
        pErr = pErr + UpdateBit(patterns,weightMatrix);
    end
    pErr = pErr/nrOfTrials;

end


%Gets weightMatrix (P x N size)
function WeightMatrix=GetWeights(patterns)
    Npatterns = size(patterns,1);
    Nbits = size(patterns,2);
    WeightMatrix = zeros(Nbits,Nbits);

    for iPattern = 1:Npatterns
        patternI = patterns(iPattern,:);
        WeightMatrix = WeightMatrix+mtimes(patternI',patternI);
    end
    WeightMatrix = WeightMatrix/Nbits;

    %remove in part B
    for iBits = 1:Nbits
        WeightMatrix(iBits,iBits)=0;
```

```
42      end
43
44  end
45
46
47  %update 1 bit asynchrounosly
48  function errorBit = UpdateBit(patterns,weight)
49      Npatterns = size(patterns,1);
50      Nbits = size(patterns,2);
51
52      pRand = fix(Npatterns*rand+1);
53      nRand = fix(Nbits*rand+1);
54
55      errorBit = 0;
56      b_nRand = patterns(pRand,:)*weight(:,nRand);  %could check if c>1 or not
57
58      sgn_bn = Sgn(b_nRand);
59      if sgn_bn ~= patterns(pRand,nRand)
60          errorBit = 1;
61      end
62  end
63
64
65
66  %sign(x) with if == 0--> =1
67  function sgn = Sgn(x)
68      sgn = sign(x);
69      if sgn == 0
70          sgn = 1;
71      end
72  end
```

Problem 2 Assignment1 Isac Nordin

isacn

September 2021

1 code

```
%Main code


%patterns
x1=[ [ -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, 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] ];
x2=[ [ -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, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1,
     1, -1, -1, -1] ];
x3=[ [ 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, -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] ];
x4=[ [ -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, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1,
    1, -1, -1] ];
```

```
x5=[ [ -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, -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] ];

%initiate model
patternBitsWidth=10;
Patterns=[x1;x2;x3;x4;x5];
weightMatrix=GetWeights(Patterns);

currentState=[[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, -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]];
%currentState=Sgn(2*rand(1,length(currentState))-1); %random state for
    experimentation

PlotPattern(currentState,patternBitsWidth)
pause(0.5)

%start simulation
while true
    oldState = currentState;
    currentState = UpdateState(currentState,weightMatrix);
    if currentState == oldState %if steady state
        break;
    end
    PlotPattern(currentState,patternBitsWidth)
    pause(0.2);
end
disp('simulation is done')

%make steady state into a openTA friendly string
openTa = PrintStateAsMatrix(currentState,patternBitsWidth)
pause(0.5)
close all

%get weightMatrix (P x N size)
function WeightMatrix = GetWeights(patterns)
    Npatterns = size(patterns,1);
    Nbits = size(patterns,2);
    WeightMatrix = zeros(Nbits,Nbits);

    for iPattern = 1:Npatterns
```

```
46          patternI = patterns(iPattern,:);
47          WeightMatrix = WeightMatrix+mtimes(patternI',patternI);
48      end
49      WeightMatrix = WeightMatrix/Nbits;
50
51      %modified hebbs rule
52      for iBits = 1:Nbits
53          WeightMatrix(iBits,iBits) = 0;
54      end
55  end
56
57  %asynchronous update of state N times
58  function newState = UpdateState(currentState,weightMatrix)
59
60      newState = currentState;
61      %N asynchronous updates in a row
62      for iBit = 1:length(currentState)
63          newState(iBit) = Sgn(weightMatrix(iBit,:)*currentState');
64          currentState(iBit) = newState(iBit);
65      end
66
67  end
68
69
70  %sign(x) with if==0 --> =1
71  function sgn = Sgn(x)
72      sgn = sign(x);
73      if sgn == 0
74          sgn = 1;
75      end
76  end
77
78  %-----code that helps answer questions-----
79
80
81  %visualizes pattern
82  function p = PlotPattern(x,widthPx)
83      xMatrix = zeros(length(x)/widthPx,widthPx);
84      for i = 1:length(x)/widthPx
85          xMatrix(i,:) = x(i*widthPx-(widthPx-1):i*widthPx);
86      end
87      imagesc(-xMatrix)
88      colormap(gray)
89  end
90
91
92  %prints answer in a openTA friendly way
93  function k = PrintStateAsMatrix(state,width)
94      k = '[';
95      for i = 1:length(state)/width
96          s = state(i*width-width+1:i*width);
97          s = string(regexprep( mat2str(s), {' '}, {','}));%to string
98          if i ~= length(state)/width
99              k = k+s + ',';
```

```
100             else
101                 k = k+s;
102             end
103         end
104         k = k+']';
105 
106 end
```

Problem3 Assignment1 Isac Nordin

isacn

September 2021

1 code

```
%Main code
N = 200;
p = 45;
T = 2*10^5;
noise_beta = 2;

%for loop to average m1 to be implemented
mAverage = 0;
averageTrial = 100;
for i = 1:averageTrial
    mAverage = mAverage + GetOrderParameter(N,p,T,noise_beta);
end
round(mAverage/averageTrial,3)

%gets orderparameter
function m1 = GetOrderParameter(N,p,T,noiseBeta)
    m1 = 0;
    patterns = sign(2*rand(p,N)-1);
    weightMatrix = GetWeights(patterns);
    x1 = patterns(1,:);
    currentState = x1;
    for iTrial=1:T
        m1 = m1 + (1/N) * currentState*x1';
        currentState=AsynchronousStochasticUpdate(currentState,weightMatrix,
            noiseBeta);
    end
    m1=m1/T;
end


%aynchrounous stochastic update of 1 bit
function newState = AsynchronousStochasticUpdate(currentState,weightMatrix,
    noiseBeta)
    nRand = floor(length(currentState)*rand+1); %update random bit
    bn = weightMatrix(nRand,:)*currentState';

    newState = currentState;
    if rand <= Pb(bn,noiseBeta)
        newState(nRand) = 1;
    else
        newState(nRand) = -1;
```

```
    end
end

%stochastic update probability
function pB = Pb(bi,noise_beta)
    pB=1/(1+exp(-2*noise_beta*bi));
end


%get WeightMatrix (P x N size)
function WeightMatrix = GetWeights(patterns)
    Npatterns = size(patterns,1);
    Nbits = size(patterns,2);
    WeightMatrix = zeros(Nbits,Nbits);

    for iPattern = 1:Npatterns
        patternI = patterns(iPattern,:);
        WeightMatrix = WeightMatrix+mtimes(patternI',patternI);
    end
    WeightMatrix = WeightMatrix/Nbits;

    %modified hebbs rule
    for iBits = 1:Nbits
        WeightMatrix(iBits,iBits) = 0;
    end
end


%sign(x) but if ==0 -> =1
function sgn = Sgn(x)
    sgn = sign(x);
    if sgn == 0
        sgn = 1;
    end
end
```