```
%% HW1 - One step error
clc, clear all;
P = [12, 24, 48, 70, 100, 120];
N = 120;
trials = 1e5;
errors = zeros(length(P),1);
possibleValues = [-1,1];

for k=1:length(P)
    for i = 1:trials
        s = possibleValues(randi(length(possibleValues), P(k), N));
        w = 1/N * (s') * s;
        w(1:1+size(w,1):end) = 0;

        m = randi(N);
        randomPattern = randi(P(k));

        x = s(randomPattern,:);
        b = w(m,:) * x';
        if b == 0
            sNext = 1;
        else
            sNext = sign(b);
        end

        if sNext ~= x(m)
            errors(k,1) = errors(k,1) + 1;
        end
    end
end
```

```
%% HW1 - Pattern recognition
clear, clc
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],[ -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, -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] ];

s1 = [[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1], [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
[-1, -1, 1, 1, 1, 1, 1, 1, -1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1,
-1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1,
1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1,
1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1,
-1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, -1, 1, 1, 1, 1, 1, 1, -1,
-1], [1, 1, 1, -1, -1, -1, -1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]];
s2 = [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1], [-1, -1,
1, 1, 1, 1, 1, 1, -1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1,
1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1],
[-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1,
```

```
-1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, 1, 1, 1, -1, -1, 1, 1,
1, -1], [-1, 1, 1, 1, -1, -1, 1, 1, 1, -1], [-1, -1, 1, 1, 1, 1, 1, 1, -1, -1], [-1,
-1, -1, 1, 1, 1, 1, -1, -1, -1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]];
s3 = [[1, -1, -1, -1, -1, 1, -1, -1, -1, -1], [1, -1, -1, 1, 1, -1, 1, -1, -1, -1],
[1, -1, 1, 1, 1, -1, 1, 1, -1, -1], [1, 1, 1, 1, -1, 1, 1, 1, 1, -1], [1, 1, 1, 1,
-1, 1, 1, 1, 1, -1], [1, 1, 1, 1, -1, 1, 1, 1, 1, -1], [1, 1, 1, 1, -1, 1, 1, 1, 1,
-1], [1, 1, 1, 1, -1, 1, 1, 1, 1, -1], [1, 1, 1, 1, -1, 1, 1, 1, 1, -1], [1, 1, 1, 1,
-1, 1, 1, 1, 1, -1], [1, 1, 1, 1, -1, 1, 1, 1, 1, -1], [1, 1, 1, 1, -1, 1, 1, 1, 1,
-1], [1, 1, 1, 1, -1, 1, 1, 1, 1, -1], [1, -1, 1, 1, 1, -1, 1, 1, -1, -1], [1, -1,
-1, 1, 1, -1, 1, -1, -1, -1], [1, -1, -1, -1, -1, 1, -1, -1, -1, -1]];

patterns = [x1;x2;x3;x4;x5];
inputP = [s1;s2;s3];

N = size(x1,2);
W = 1/N * (patterns') * patterns;
W(1:1+size(W,1):end) = 0;

for input = 1:size(inputP,1)
    notSteady = true;
    s = inputP(input,:);
    while notSteady
        for i = 1:N
            b = W(i,:) * s';
                if b == 0
                    sNext = 1;
                else
                    sNext = sign(b);
                end
            s(i) = sNext;
        end

        for pattern = 1:size(patterns,1)
            error = sum ( s ~= patterns(pattern,:) );
            if error == 0
                notSteady = false;
                disp("Converged to pattern: " + pattern)
            end
            if error == N
                notSteady = false;
                disp("Converged to pattern: " + -pattern)
            end
        end
    end
end
```

```
%% Boolean function
clear all, clc
N=2; %3, 4, 5

nTrails = 1e4;
nEpochs = 20;
nu = 0.05;
counter = 0;
ifAdd = 0;
used_bool = zeros(2^2^N,2^N);
possibleValues = [-1,1];

% Creates boolean input for N dim
s=0;
for i = 0:N-1
    s = s + 2^(i);
end
boolean_inputs = (dec2bin(0:s)-'0')';

for trail = 1:nTrails
    boolean_output = possibleValues(randi(length(possibleValues), 2^N, 1));
    if ~ismember(boolean_output, used_bool, 'rows')
        w = randn(1,N)/sqrt(N);
        theta = zeros(1,2^N);

        for epoch = 1:nEpochs
            total_error = 0;
            for i = 1:2^N
                O = sign(sum(w*(boolean_inputs(:,i))) - theta(i));
                error = boolean_output(i) - O;

                dw = nu*(boolean_output(i) - O) * boolean_inputs(:,i)';
                w = w + dw;
                dtheta = -nu *(boolean_output(i) - O);
                theta = theta + dtheta;

                total_error = total_error + abs(error);
            end
            if total_error == 0
                counter = counter +1;
                break
            end
        end
        ifAdd = ifAdd + 1;
        used_bool(ifAdd,:) = boolean_output;
    end
end
```

Boolean functions

FFR135 Artificial neural networks

Axel Johansson
axejoh@student.chalmers.se

September 19, 2022

Solution proposal

There are 2^{2^n} Boolean functions, where n is the dimension. Since we do 10 000 trials in this lab we only check all functions for dimension 2 and 3. Therefore 4 and 5 there are plenty more Boolean functions than this and thus the results vary to a larger degree for these numbers, especially for dimension 4. For dimensions 2 to 3 we check every function so the number of linearly separable functions increase. However since there are so many functions for dimension 4 and 5 that we don't check all with the 10 000 trials. For previous dimensions the predicted number we would find with the algorithm would be equal to the number that exists. In tabular 1 it seems like the algorithm gave just that. To understand the results in dimension 4 and 5 one has to understand whats expected. For these dimensions a much better predictor is to take the expected value for one trial and multiply by the number of trials. For dimension 5 the expected value is $94572/(2^{2^5}) = 0{,}000022$. For 10 000 trials this becomes around 22% to find one linearly separable function which also is reflected in tabular 1 with a 0. Conclusion is that it seems to work, especially after doing multiple runs and getting 1 separable function around every 5th try.

Table 1: The number of linearly separable boolean function for each dimension.

Dimension	# linearly separable
2	14
3	104
4	277
5	0

It also seems as the algorithm doesn't always find the same number of separable functions even for the smaller dimensions. This could happen if the initial line that is randomized is "too bad" so that the 20 training epochs isn't enough training. This problem seems to vanish as the number of epochs increase but slows down the program.