FFR135, Artificial Neural Networks
Home Problem 2

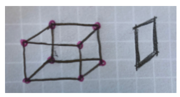
6 oktober 2019

Ella Guiladi
930509-0822
guiladi@student.chalmers.se

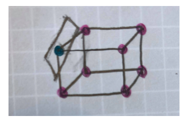
1 3-dimensional Boolean functions

In this report a blue ball indicates $t^{(\mu)} = 1$, a pink ball indicates $t^{(\mu)} = 0$ and k refers to the blue balls, i.e the number of ones. Since there are 2^3 unique combinations, one analyse the cases for k=1,2,3,4,5,6,7,8. Instead of looking at the cases for k=5,6,7,8 it is possible to duplicate the result for the k=0,1,2,3 cases, to get the number of linearly separable functions, due to symmetry.

Figure (1a) presents the symmetry when k=0, which results in 1 linearly separable function, since the boundary plane only can be positioned outside of the cube. In figure (1b) the symmetry for k=1 is presented, which shows 1 of the 8 linearly separable functions. The blue ball can be in each of the eight corners, resulting in 8 linearly separable functions.



(a)



(b)

Figure 1: Symmetries for k=0 in a) and and k=1 in b)

The case for k=2 is presented in figure (2). The linearly separability for k=2 is displayed only in cube (1) in figure (2), where the boundary plane for one case is presented. From the symmetries there will be 12 linearly separable functions due to the boundary planes positioning on each side of the cube.

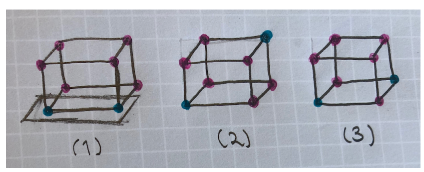


Figure 2: Symmetries for k=2

When k=3, figure (3) shows that only cube (1) is linearly separable. For cube(1) there are 4 different combinations on each side and there are 6 different sides, resulting in $4 \cdot 6 = 24$ linearly separable functions.

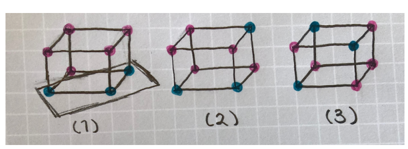


Figure 3: Symmetries for k=3

When k=4 in figure (4), the result shows that only cube (1) and (2) are linearly separable. Cube (1) have 6 linearly separable functions, due to having 6 sides of the cube. Cube (2) have 8 linearly separable functions since the boundary plane can cut the cube in half from each corner.

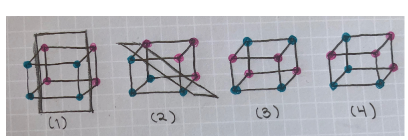


Figure 4: Symmetries for k=4

Finally, the total number of linearly separable functions will thereby be: $2 \cdot 1 + 2 \cdot 8 + 2 \cdot 12 + 2 \cdot 24 + 6 + 8 = 104$

1 Linear separability of 4-dimensional Boolean functions

```
clear
clc
learningRate=0.02;
numberUpdates=10^5;
repetitions=10;

matrixOfPatterns=csvread('input_data_numeric.csv');

A = [1, 1, 1, 1, 1, 1, 1, -1, 1, -1, 1, -1, -1, -1, -1, -1];
B = [-1, -1, 1, 1, 1, -1, -1, -1, -1, 1, 1, -1, 1, 1, 1, -1];
C = [1, 1, 1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1, -1, 1, -1];
D = [-1, -1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1];
E = [1, 1, 1, -1, 1, 1, -1, 1, 1, 1, -1, 1, -1, -1, -1, 1];
F = [1, -1, 1, -1, -1, 1, -1, -1, -1, -1, -1, -1, 1, -1, -1, 1];
targetPattern= A'; %change inputpatterns here

output=zeros(16,1);
weightsNew=[];
thresholdNew=[];
weights=[];

for l=1:repetitions

for i=1:4
weights(i)= -0.2 + (0.2+0.2).*rand(1,1);
end
threshold = -1 + (1-(-1)).*rand(1,1);

for j=1:numberUpdates
    my = randi([1,16]);

theSumInTheOutput=0;
for k=1:4
    summationOutput=weights(k).*matrixOfPatterns(my,k);
    theSumInTheOutput=theSumInTheOutput+summationOutput;
end

output(my)=tanh((1/2)*(-threshold+theSumInTheOutput));

for n=1:4
weightsNew(n)=weights(n)+learningRate.*(targetPattern(my)-output(my)).*(1-(↩
     tanh((1/2)*(-threshold+theSumInTheOutput))^2))*(1/2)*matrixOfPatterns(↩
     my,n);
end

thresholdNew=threshold-learningRate*(targetPattern(my)-output(my)).*(1-(↩
     tanh((1/2)*(-threshold+theSumInTheOutput))^2))*(1/2);

threshold=thresholdNew;
weights=weightsNew;
end
if sign(output)==targetPattern
    disp('linearly separable')
else
    disp('not linearly separable')
end
end
```

2 Two-layer perceptron

```
clear
clc
trainingSet = csvread('training_set.csv');
validationSet = csvread('validation_set.csv');
learningRate=0.01;
numberOfUpdates=10^6;
patternsValidationSet=5000;
M1=15;
M2=25;

firstLayer = [];
secondLayer = [];
output=0;

outputError=[];
secondLayerError=[];
firstLayerError=[];

firstLayerValidation = zeros(M1);
secondLayerValidation = zeros(M2);
outputValidation=0;

%initializing thresholds
firstThreshold =  normrnd(0,1,[1,M1]);
secondThreshold =  normrnd(0,1,[1,M2]);
outputThreshold =  normrnd(0,1,[1,1]);

%initializing weights
inputWeight= normrnd(0,1,[M1, 2]);
hiddenWeight= normrnd(0,1,[M2,M1]);
outputWeight= normrnd(0,1,[1,M2]);


%training
for i=1:numberOfUpdates
    my = randi([1, 10000]);

    %forward propagation
    firstLayerSum=[];
    for s = 1:M1
        firstLayerSum(s)=sum(inputWeight(s,:).*trainingSet(my,1:2));
        firstLayer(s)=tanh(-firstThreshold(s)+firstLayerSum(s));
    end

    secondLayerSum=[];
    for q=1:M2
        secondLayerSum(q)=sum(hiddenWeight(q,:).*firstLayer);
        secondLayer(q) = tanh(-secondThreshold(q)+secondLayerSum(q));
    end

    outputSum=sum(outputWeight.*secondLayer);
    output=tanh(-outputThreshold+outputSum);


    %Backpropagation
    %errors are updates from right to left
    outputError=(trainingSet(my,3)-output)*(1-(tanh(-outputThreshold+↩
         outputSum))^2);

    for c=1:M2
        secondLayerError(c)=outputError*outputWeight(c)*(1-(tanh(-↩
             secondThreshold(c)+secondLayerSum(c)))^2);
    end
```

```
    for d=1:M1
        firstLayerError(d)=sum(secondLayerError.*hiddenWeight(:,d)')*(1-(↩
             tanh(-firstThreshold(d)+firstLayerSum(d))^2));
    end


    %weight update
    %neurons are updated from left to right
    updatedInputWeight= [];
    updatedHiddenWeight = [];
    updatedOutputWeight = [];

    for u=1:M1
        for v=1:2
            updatedInputWeight(u,v)= inputWeight(u,v)+learningRate*↩
                 firstLayerError(u)*trainingSet(my,v);
        end
    end


    for z=1:M2
        for y=1:M1
            updatedHiddenWeight(z,y)= hiddenWeight(z,y)+learningRate*↩
                 secondLayerError(z)*firstLayer(y);
        end
    end

    for h=1:M2
        updatedOutputWeight(h)= outputWeight(h)+learningRate*outputError*↩
             secondLayer(h);
    end

    %updating thresholds
    newFirstThreshold = [];
    newSecondThreshold = [];

    for bc=1:M1
        firstThreshold(bc)=firstThreshold(bc)-learningRate*firstLayerError(↩
             bc);
    end

    for de=1:M2
        newSecondThreshold(de)=secondThreshold(de)-learningRate*↩
             secondLayerError(de);
    end

    newOutputThreshold = outputThreshold-learningRate*outputError;

    %write over old weights and thresholds
    inputWeight=updatedInputWeight;
    hiddenWeight=updatedHiddenWeight;
    outputWeight=updatedOutputWeight;

    newFirstThreshold=firstThreshold;
    secondThreshold=newSecondThreshold;
    outputThreshold=newOutputThreshold;

end

%validation
sumOfValidation=0;
for j=1:patternsValidationSet

    theSumInTheOutput11Validation=[];
    for pq = 1:M1
        theSumInTheOutput1Validation=0;
        for rs=1:2
```

```
            summationOutput1Validation=inputWeight(pq,rs)*validationSet(j,↩
                 rs);
            theSumInTheOutput1Validation=theSumInTheOutput1Validation+↩
                 summationOutput1Validation;
        end
        theSumInTheOutput11Validation(pq)=theSumInTheOutput1Validation;
        firstLayerValidation(j,pq)=tanh(-firstThreshold(pq)+↩
             theSumInTheOutput11Validation(pq));
    end

    theSumInTheOutput22Validation=[];
    for tu=1:M2
        theSumInTheOutput2Validation=0;
        for vx=1:M1
            summationOutput2Validation=hiddenWeight(tu,vx)*↩
                 firstLayerValidation(j,vx);
            theSumInTheOutput2Validation=theSumInTheOutput2Validation+↩
                 summationOutput2Validation;
        end
        theSumInTheOutput22Validation(tu)=theSumInTheOutput2Validation;
        secondLayerValidation(j,tu) = tanh(-secondThreshold(tu)+↩
             theSumInTheOutput22Validation(tu));
    end

    theSumInTheOutput3Validation=0;
    for f=1:M2
        summationOutput3Validation=outputWeight(f)*secondLayerValidation(j,↩
             f);
        theSumInTheOutput3Validation=theSumInTheOutput3Validation+↩
             summationOutput3Validation;
    end

    outputValidation(j)=tanh(-outputThreshold+theSumInTheOutput3Validation)↩
         ;


    partSumOfValidation=abs(sign(outputValidation(j))-validationSet(j,3));
    sumOfValidation=sumOfValidation+partSumOfValidation;

end

C = ((1/(2*patternsValidationSet))*(sumOfValidation))


disp('The run is done!')

csvwrite('w1.csv',inputWeight);
csvwrite('w2.csv',hiddenWeight);
csvwrite('w3.csv',outputWeight');
csvwrite('t1.csv',firstThreshold');
csvwrite('t2.csv',secondThreshold');
csvwrite('t3.csv',outputThreshold);
```