3-dimensional Boolean functions

David Tonderski (davton)

I am using the following notation: (10000001) represents the boolean function t such that that $t^{(\mu)} = 1$ for $\mu \in \{1, 8\}$, $t^{(\mu)} = 0$ for $\mu \in \{2, 3, 4, 5, 6, 7\}$, where the μ values are visualised in figure 1. As in the task description on OpenTA, this is also represented by the ball $\mu = 1$ and the ball $\mu = 8$ being black.

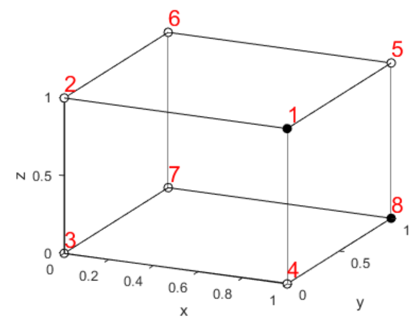

Figure 1: The location of μ in 3d-space and the function (10000001). For example, $\mu = 1$ represents the point $(1, 0, 1)$. Black balls indicate $t^{(\mu)} = 0$, white balls indicate $t^{(\mu)} = 1$.

Next, we split up the problem into looking at cases where the functions map exactly k input patterns are matched to 1. We know that the number of such functions is $N_k = \binom{8}{k}$. These functions can be grouped into symmetries whose "cubes" can be mapped onto each other by reflection or rotation. Let us call the number of functions belonging to such a symmetry as N_k^j with $j = 1, 2, ..., J$, where J is the number of symmetries for a given k. Next, let us call the number of linearly separable functions that map k patterns to 1 as n_k. By symmetry, we have $n_k = n_{(8-k)}$ (switch colors). Therefore, we only need to analyze $k \in \{0, 1, 2, 3, 4\}$.

Let's begin with $k = 0$. Trivially, there is only 1 such function, and it is linearly separable: $n_0 = n_8 = 1$.

Next, we analyze $k = 1$. This is the same as choosing one corner. Therefore, we have only one symmetry ($J = 1$), which is also linearly separable, so $n_1 = n_7 = N_1^1 = 8$.

Next, we have $k = 2$. Here, we have $J = 3$: $j = 1$ is choosing the black balls to be along one edge: $N_2^1 = N_{edges} = 12$, $j = 2$ is choosing the black balls to be along a face diagonal: $N_2^2 = 2N_{faces} = 12$, and $j = 3$ is choosing the black balls to be along a cube diagonal: $N_2^3 = 4$. We double check: $\sum_{j=1}^{J=3} N_2^j = 12 + 12 + 4 = 28 = \binom{8}{2}$. Only the symmetry $j = 1$ is linearly separable, so $n_2 = n_6 = N_2^1 = 12$.

Next, we have $k = 3$. Here, the symmetries are difficult to explain geometrically, so I will explain the linearly separable ones, but only give examples of the others. The only linearly separable symmetry is when we choose all three black balls to be on a single face. We have 6 faces, and each face has $\binom{4}{3} = 4$ ways to choose three balls, so we have $N_3^1 = 4 \cdot 6 = 24$. Next, we have the second symmetry (11000001) with $N_3^2 = 24$, and the third symmetry (01011000) with $N_3^3 = 8$. Double check: $\sum_{j=1}^{J=3} N_3^j = 24 + 24 + 8 = 56 = \binom{8}{3}$. We have $n_3 = n_5 = N_3^1 = 24$.

Lastly, we have $k = 4$. Again, the symmetries are difficult to explain geometrically. There are two linearly separable ones: firstly, when the black balls are along edges belonging to one face (e.g. (11110000)), with $N_4^1 = N_{faces} = 6$, and secondly, when the black balls are along the edges that are connected to one common corner (e.g. (11100100)), with $N_4^2 = N_{corners} = 8$. There are four other symmetries: (01110001) with $N_4^3 = 24$, (11000011) (here, the black balls are chosen along opposing edges) with $N_4^4 = 6$, (01011010) with $N_4^5 = 2$, and (01111000) with $N_4^6 = 24$. Double check: $\sum_{j=1}^{J=6} N_3^j = 6 + 8 + 24 + 6 + 2 + 24 = 70 = \binom{8}{4}$. We have $n_4 = N_4^1 + N_4^2 = 14$.

Finally, we have the number of linearly separable functions

$$n = \sum_{k=0}^{k=8} n_k = 1 + 8 + 12 + 24 + 14 + 24 + 12 + 8 + 1 = 104. \tag{1}$$

LINEAR SEPARABILITY OF 4-DIMENSIONAL BOOLEAN FUNCTIONS

Table of Contents

Load data

```
clear
inputData = load("input_data_numeric.csv")';
targetDataA = [1, -1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, 1];
targetDataB = [1, -1, 1, 1, -1, 1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1];
targetDataC = [-1, 1, -1, 1, -1, 1, -1, 1, -1, 1, 1, 1, -1, -1, 1,
 -1];
targetDataD = [1, 1, 1, -1, 1, 1, -1, 1, 1, 1, -1, 1, -1, -1, -1, 1];
targetDataE = [1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, 1, 1, -1, -1,
 -1];
targetDataF = [1, 1, 1, -1, 1, 1, 1, 1, -1, -1, -1, -1, 1, 1, -1, -1];
targetData = [targetDataA; targetDataB; targetDataC; ...
    targetDataD; targetDataE; targetDataF];
```

Check linear separability

```
T = 1e5;
learningRate = 0.02;
alphabet = ['A', 'B', 'C', 'D', 'E', 'F'];
for i = 1:6
    currentTargetData = targetData(i,:);
    threshold = rand();
    weights = randomWithinRange([1 4], -0.2, 0.2);
    energies = zeros(1,T);
    for t = 1:T
        patternNumber = randi([1 16]);
        input = inputData(2:5, patternNumber);
        output = calculateOutput(input, weights, threshold);
        target = currentTargetData(patternNumber);
        b = -threshold + weights*input;
        delta = learningRate*(target - output)*gPrime(b);
        deltaWeights = delta*input';
        weights = weights + deltaWeights;
        deltaThreshold = -delta;
        threshold = threshold + deltaThreshold;
        outputs = calculateOutputs(inputData(2:5,:), weights,
 threshold);
```

```
        energies(t) = energyFunction(outputs, currentTargetData);
    end
    success = (all(sign(outputs) == currentTargetData));
    if success
        disp(['Function ', alphabet(i), ' is linearly separable.'])
    else
        disp(['Function ', alphabet(i), ' is not linearly
 separable.'])
    end
end
```

```
Function A is linearly separable.
Function B is not linearly separable.
Function C is not linearly separable.
Function D is not linearly separable.
Function E is linearly separable.
Function F is linearly separable.
```

Functions

```
disp('')
function r = randomWithinRange(size, min, max)
    r = (max - min).*rand(size) + min;
end

function g = g(x)
    g = tanh(x);
end

function gPrime = gPrime(x)
    gPrime = 1-(tanh(x)).^2;
end

function output = calculateOutput(input, weights, threshold)
    output = g(1./2.*(-threshold + weights*input));
end

function outputs = calculateOutputs(inputs, weights, threshold)
    numberOfInputs = size(inputs,2);
    outputs = zeros(1, numberOfInputs);
    for i = 1:numberOfInputs
        input = inputs(:, i);
        outputs(i) = calculateOutput(input, weights, threshold);
    end
end

function energy = energyFunction(outputs, targets)
    energy = 1/2*sum((outputs-targets).^2);
end
```

Published with MATLAB® R2020a

TWO-LAYER PERCEPTRON

Table of Contents

Load data

```
clear;
trainingSet = load('training_set.csv')';
validationSet = load('validation_set.csv')';
```

Initialize variables and constants

```
M1 = 8;
M2 = 6;
learningRate = 0.02;

w1 = randomWithinRange([M1, 2], -0.2, 0.2);
t1 = zeros(M1, 1);
w2 =  randomWithinRange([M2, M1], -0.2, 0.2);
t2 = zeros(M2, 1);
w3 = randomWithinRange([1, M2], -0.2, 0.2);
t3 = 0;

% Save validation error values every tFrequency points
tFrequency = 1e4;
errorCondition = 0.12;
errorArray = [];
t = 0;
```

Stochastic gradient descent until C < 12%

```
while true
    t = t+1;
    patternNumber = randi([1 length(validationSet)]);
    input = trainingSet(1:2, patternNumber);
    target = trainingSet(3, patternNumber);
    V1 = g(-t1 + w1*input);
    V2 = g(-t2 + w2*V1);
    output = calculateOutput(input, w1, t1, w2, t2, w3, t3);
    delta3 = gPrime(-t3 + w3*V2)*(target-output);
    delta2 = gPrime(-t2 + w2*V1).*(w3'*delta3);
    delta1= gPrime(-t1 + w1*input).*(w2'*delta2);
```

```
    if mod(t-1, tFrequency) == 0
        error = validationError(validationSet, w1, t1, w2, t2, w3,
 t3);
        errorArray((t-1)/tFrequency+1) = error;
        if error < errorCondition
            break
        end
    end
    w1 = w1 + learningRate*delta1*input';
    w2 = w2 + learningRate*delta2*V1';
    w3 = w3 + learningRate*delta3*V2';
    t1 = t1 - learningRate*delta1;
    t2 = t2 - learningRate*delta2;
    t3 = t3 - learningRate*delta3;
end
```

Save data

```
csvwrite('w1.csv', w1)
csvwrite('w2.csv', w2)
csvwrite('w3.csv', w3')
csvwrite('t1.csv', t1)
csvwrite('t2.csv', t2)
csvwrite('t3.csv', t3)
```

Functions

```
function error = validationError(validationSet, w1, t1, w2, t2, w3,
 t3)
    pVal = length(validationSet);
    errorSum = 0;
    for i = 1:pVal
        input = validationSet(1:2,i);
        output = calculateOutput(input, w1, t1, w2, t2, w3, t3);
        target = validationSet(3,i);
        errorSum = errorSum + abs(sign(output)-target);
    end
    error = errorSum./(2.*pVal);
end

function g = g(x)
    g = tanh(x);
end

function gPrime = gPrime(x)
    gPrime = 1-(tanh(x)).^2;
end

function output = calculateOutput(input, w1, t1, w2, t2, w3, t3)
    V1 = g(-t1 + w1*input);
    V2 = g(-t2 + w2*V1);
    output = g(-t3 + w3*V2);
end
```

```
function r = randomWithinRange(size, min, max)
    r = (max - min).*rand(size) + min;
end
```

Published with MATLAB® R2020a