1. *How many linearly separable 3-dimensional Boolean functions are there? To answer this question, use graphical representation of the different functions, and symmetries to reduce the number of cases to be considered. You must also upload a one-page pdf file with the explanation of how you arrived at the answer. Online sources are not accepted as a valid answer.*

**k=0 (1):** we have 1 possible function and it is linearly separable.

**k=1 (8):** we have 8 possible functions, and only one symmetry. They are all linearly separable.

**k=2 (12):** we have 3 symmetries as shown below, only the first one is linearly separable. There are 12 linearly separable functions in this case.



**k=3 (24):** we have 4 symmetries as shown below. Only the first set is linearly separable, and there are 24 different functions in this symmetry, so we have 24 linearly separable functions in this case.



**k=4 (14):** we have 7 symmetries as shown below. In the 1st set we have 6 different functions and they are linearly separable. The 2nd set is not linearly separable. The 3rd set is linearly separable and there are 8 different functions. The 4th, 5th, 6th, and 7th sets are not linearly separable.





**k=5 (24):** we have only one symmetry that is linearly separable (the first one). There are 24 functions in this set.



**k=6 (12):** we have two linearly separable symmetries, the 1st one below. There are 12 functions in this case.



**k=7 (8):** we have only one symmetry, with 8 possible functions, and it is linearly separable.



**k=8 (1):** we have only one possible function, which is linearly separable.



Linear separability of 4-dimensional Boolean functions (2020)

X = readmatrix('input\_data\_numeric.csv');

X(:,1)=[];

boolean\_functions = readmatrix('boolean\_functions.txt');

t = boolean\_functions(6,:);

n = .02;

W = -.2 + .4.\*rand(1,4);

T = -1 + 2\*rand;

converged = 0;

H = zeros(1,10^5);

for iRun = 1:10

iLearned = 1;

O = zeros(1,16);

sigO = zeros(1,16);

while converged == 0 && iLearned < 10^5

% calc output for all patterns

for mu=1:16

dotProduct = dot(X(mu,:),W);

O(mu) = tanh(dotProduct - T);

if O(mu) >= 0

sigO(mu) = 1;

else

sigO(mu) = -1;

end

H(iLearned) = H(iLearned) + (t(mu)-O(mu))^2;

end

H(iLearned) = H(iLearned)/2;

% check for convergence

if isequal(sigO,t)

converged = 1;

break

end

% pick random pattern and update weights, thresh

muRand = randi(16,1);

dotProduct = dot(X(muRand,:),W);

gPrime = 1 - tanh(dotProduct-T)^2;

gradient = gPrime \* (t(muRand)-O(muRand));

for i=1:4

W(i) = W(i) + n\*gradient\*X(muRand,i);

end

T = T + -n\*gradient;

iLearned = iLearned + 1;

end

end

H(H==0) = [];

converged

plot(H(1:end))

Two-layer perceptron (2020)

clear all

M1 = 8;

M2 = 4;

step = .02;

numRuns = 5000;

% training variables

train = readmatrix('training\_set.csv');

X = train(:,1:2);

t = train(:,3);

pVal = length(X);

w{1} = -.2 + .4.\*rand(M1,2);

w{2} = -.2 + .4.\*rand(M2,M1);

w{3} = -.2 + .4.\*rand(1,M2);

theta{1} = zeros(M1,1);

theta{2} = zeros(M2,1);

theta{3} = 0;

B{1} = zeros(M1,pVal);

B{2} = zeros(M2,pVal);

B{3} = zeros(1,pVal);

V{1} = zeros(M1,pVal);

V{2} = zeros(M2,pVal);

V{3} = zeros(1,pVal);

sigO = zeros(1,pVal);

err{1} = zeros(M1);

err{2} = zeros(M2);

err{3} = zeros(1);

C = zeros(1,numRuns);

% testing variables

test = readmatrix('validation\_set.csv');

testInput = test(:,1:2);

testTarget = test(:,3);

pValTest = length(testInput);

wGood{1} = zeros(M1,2);

wGood{2} = zeros(M2,M1);

wGood{3} = zeros(1,M2);

thetaGood{1} = theta{1};

thetaGood{2} = theta{2};

thetaGOod{3} = theta{3};

BTest{1} = B{1};

BTest{2} = B{2};

BTest{3} = B{3};

VTest{1} = V{1};

VTest{2} = V{2};

VTest{3} = V{3};

sigOTest = sigO;

CTest = zeros(1,numRuns);

numIter = 0;

for iRun=1:numRuns

% calc Bs, Vs, and final output sigO

for mu=1:pVal

B{1}(:,mu) = w{1} \* X(mu,:)' - theta{1};

V{1}(:,mu) = tanh(B{1}(:,mu));

for L=2:3

B{L}(:,mu) = w{L} \* V{L-1}(:,mu) - theta{L};

V{L}(:,mu) = tanh(B{L}(:,mu));

end

if V{3}(:,mu) >= 0

sigO(:,mu) = 1;

else

sigO(:,mu) = -1;

end

C(iRun) = C(iRun) + abs(sigO(:,mu) - t(mu));

end

C(iRun) = C(iRun)/(2\*pVal);

for numFed=1:pVal

numIter = numIter + 1;

iRand = randi(pVal,1);

% calc local fields, Vs

B{1}(:,iRand) = w{1} \* X(iRand,:)' - theta{1};

V{1}(:,iRand) = tanh(B{1}(:,iRand));

for L=2:3

B{L}(:,iRand) = w{L} \* V{L-1}(:,iRand) - theta{L};

V{L}(:,iRand) = tanh(B{L}(:,iRand));

end

% calc errors

err{3} = (t(iRand)-V{3}(:,iRand)) \* (1 - tanh(B{3}(:,iRand))^2);

for L=flip(1:2)

err{L} = w{L+1}' \* err{L+1} .\* (1 - tanh(B{L}(:,iRand)).^2);

end

% update weights and biases

w{1} = w{1} + step \* err{1} \* X(iRand,:);

w{2} = w{2} + step \* err{2} \* V{1}(:,iRand)';

w{3} = w{3} + step \* err{3} \* V{2}(:,iRand)';

theta{1} = theta{1} - step \* err{1};

theta{2} = theta{2} - step \* err{2};

theta{3} = theta{3} - step \* err{3};

end

% calc C for test data

for i=1:pValTest

BTest{1}(:,i) = w{1} \* testInput(i,:)' - theta{1};

VTest{1}(:,i) = tanh(BTest{1}(:,i));

for L=2:3

BTest{L}(:,i) = w{L} \* VTest{L-1}(:,i) - theta{L};

VTest{L}(:,i) = tanh(BTest{L}(:,i));

end

if VTest{3}(:,i) >= 0

sigOTest(:,i) = 1;

else

sigOTest(:,i) = -1;

end

CTest(iRun) = CTest(iRun) + abs(sigOTest(:,i) - testTarget(i));

end

CTest(iRun) = CTest(iRun)/(2\*pValTest);

if mod(iRun,10) == 0

plot(CTest)

drawnow

end

if CTest(iRun) < .12 % store good weights and biases

for L=1:3

wGood{L} = w{L};

thetaGood{L} = theta{L};

end

end

if CTest(iRun) < .115 % stopping criteria

break

end

end