The numbers of linearly separable Boolean functions found for n = 2,3,4,5 dimensions are shown in Table 1.

number of dimensions	number of linearly separable Boolean functions found
2	12
3	104
4	265
5	0

Table 1: The numbers of linearly separable Boolean functions found with different numbers of dimensions.

It should be noted that among these data, there are 16 Boolean functions in two dimensions and 256 Boolean functions in three dimensions, both of which are far less than 10,000.

So when n=2, the number of linearly separable Boolean functions is 2, and the proportion among the number of all Boolean functions is 6/7.

When n=3, the number of linearly separable Boolean functions is 104, and this value is 13/32.

When n=4, there are a total of $2^{16} = 65536$ Boolean functions available, and the number of samples is 10000, so it can be inferred that about 1737 functions are linearly separable.

When n=5, the number of linearly separable Boolean functions is $2^{25} = 4294967296$, which is much larger than the number of samples. At this time, the number of linearly separable functions is 0, because the proportion of linearly separable functions in the number of all Boolean functions is very small.

At the same time, when n=3, the number of linearly separable Boolean functions sometimes changes slightly, which may be caused by too few iterations or too low learning rate. The number of iterations can be increased to reduce the probability of misjudgment.

```
clear;
x = [0 0 ; 0 1 ; 1 0 ; 1 1];

t = zeros(4,16);
for i = 1:16
    t(:,i) = reshape(str2num(transpose(dec2bin(i-1,4))) ,4,1);
end
t(t==0) = -1;

weightVectorVariance = sqrt(1/2);
weightVectorMean = 0;
learningRate = 0.05;
thresholdValue = 0;
trainingEpoch = 20;
trainingResult = zeros(4,1);
resultNumber = zeros(2,1);

weightVector = weightVectorVariance.*randn(2,1) + weightVectorMean;

for i = 1:16
    for j = 1:trainingEpoch
        for k = 1:4
            output = O(x(k,:), weightVector, thresholdValue);
            deltaWeightVector = transpose(learningRate * (t(k,i) - output ) *
x(k,:));
            deltaThresholdValue = -learningRate * (t(k,i) - output );
            weightVector = weightVector + deltaWeightVector;
            thresholdValue = thresholdValue + deltaThresholdValue;
        end
    end
    for l = 1:4
        trainingResult(l) = O(x(l,:), weightVector, thresholdValue);
    end
    if trainingResult == t(:,i)
        resultNumber(1) = resultNumber(1) + 1;
    else
        resultNumber(2) = resultNumber(2) + 1;
    end
end
disp(resultNumber(1));
```

```
clear;
x = zeros(8,3);

for i = 1:8
    x(i,:) = reshape(str2num(transpose(dec2bin(i-1,3))) ,1,3);
end

t = zeros(8,256);
```

```
for i = 1:256
    t(:,i) = reshape(str2num(transpose(dec2bin(i-1,8))) ,8,1);
end
t(t==0) = -1;

weightVectorVariance = sqrt(1/2);
weightVectorMean = 0;
learningRate = 0.05;
thresholdValue = 0;
trainingEpoch = 20;
trainingResult = zeros(8,1);
resultNumber = zeros(2,1);

weightVector = weightVectorVariance.*randn(3,1) + weightVectorMean;

for i = 1:256
    for j = 1:trainingEpoch
        for k = 1:8
            output = O(x(k,:), weightVector, thresholdValue);
            deltaWeightVector = transpose(learningRate * (t(k,i) - output ) *
x(k,:));
            deltaThresholdValue = -learningRate * (t(k,i) - output );
            weightVector = weightVector + deltaWeightVector;
            thresholdValue = thresholdValue + deltaThresholdValue;
        end
    end
    for l = 1:8
        trainingResult(l) = O(x(l,:), weightVector, thresholdValue);
    end
    if trainingResult == t(:,i)
        resultNumber(1) = resultNumber(1) + 1;
    else
        resultNumber(2) = resultNumber(2) + 1;
    end
end
disp(resultNumber(1));
```

```
clear;
x = zeros(16,4);

for i = 1:16
    x(i,:) = reshape(str2num(transpose(dec2bin(i-1,4))) ,1,4);
end

t = zeros(16,10000);
tStore = randperm(65536, 10000);
for i = 1:10000
    t(:,i) = reshape(str2num(transpose(dec2bin(tStore(i)-1,16))) ,16,1);
end
```

```
t(t==0) = -1;

weightVectorVariance = sqrt(1/2);
weightVectorMean = 0;
learningRate = 0.05;
thresholdValue = 0;
trainingEpoch = 20;
trainingResult = zeros(16,1);
resultNumber = zeros(2,1);

weightVector = weightVectorVariance.*randn(4,1) + weightVectorMean;

for i = 1:10000
    for j = 1:trainingEpoch
        for k = 1:16
            output = O(x(k,:), weightVector, thresholdValue);
            deltaWeightVector = transpose(learningRate * (t(k,i) - output ) *
x(k,:));
            deltaThresholdValue = -learningRate * (t(k,i) - output );
            weightVector = weightVector + deltaWeightVector;
            thresholdValue = thresholdValue + deltaThresholdValue;
        end
    end
    for l = 1:16
        trainingResult(l) = O(x(l,:), weightVector, thresholdValue);
    end
    if trainingResult == t(:,i)
        resultNumber(1) = resultNumber(1) + 1;
    else
        resultNumber(2) = resultNumber(2) + 1;
    end
end
disp(resultNumber(1));
```

```
clear;
x = zeros(32,5);

for i = 1:32
    x(i,:) = reshape(str2num(transpose(dec2bin(i-1,5))) ,1,5);
end

t = zeros(32,10000);
tStore = randperm(4294967296, 10000);
for i = 1:10000
    t(:,i) = reshape(str2num(transpose(dec2bin(tStore(i)-1,32))) ,32,1);
end
t(t==0) = -1;

weightVectorVariance = sqrt(1/2);
```

```
weightVectorMean = 0;
learningRate = 0.05;
thresholdValue = 0;
trainingEpoch = 20;
trainingResult = zeros(32,1);
resultNumber = zeros(2,1);

weightVector = weightVectorVariance.*randn(5,1) + weightVectorMean;

for i = 1:10000
    for j = 1:trainingEpoch
        for k = 1:32
            output = O(x(k,:), weightVector, thresholdValue);
            deltaWeightVector = transpose(learningRate * (t(k,i) - output ) *
x(k,:));
            deltaThresholdValue = -learningRate * (t(k,i) - output );
            weightVector = weightVector + deltaWeightVector;
            thresholdValue = thresholdValue + deltaThresholdValue;
        end
    end
    for l = 1:32
        trainingResult(l) = O(x(l,:), weightVector, thresholdValue);
    end
    if trainingResult == t(:,i)
        resultNumber(1) = resultNumber(1) + 1;
    else
        resultNumber(2) = resultNumber(2) + 1;
    end
end
disp(resultNumber(1));
```

```
function output = O(x, weightVector, thresholdValue)

if (x * weightVector - thresholdValue) == 0
    output = 1;
else
    output = sign(x * weightVector - thresholdValue);
end

end
```