Hebbian learning limitations

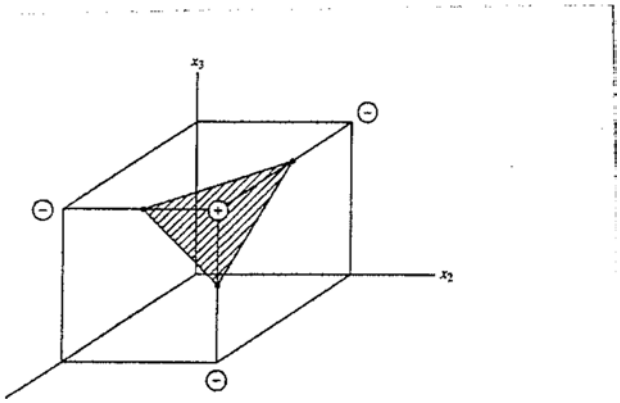
In the previous example the network *did learn* to perform the logic "AND" function for bipolar inputs and targets. However, this simple form of learning has several limitations.

Hebbian learning limitations for binary data

Consider the following set of input patterns and targets encoded in binary representation:

x1	x2	x3		target
1	1	1	->	1
1	1	0	->	0
1	0	1	->	0
0	1	1	->	0

The figure below shows that the problem is linearly separable:



However, we realize that no learning will take place for patterns 2, 3 and 4, since t = 0 for all of them. So, for the network to learn from all the training patterns, we need to re-encode the target values in bipolar format (-1 and +1).

Doing this and including a 4th column for the bias input:

Inputs					changes				weights			
x1	x2	x3	1	t	w1	w2	w3	b	w1	w2	w3	b
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	1	-1	-1	-1	0	-1	0	0	1	0
1	0	1	1	-1	-1	0	-1	-1	-1	0	0	-1
0	1	1	1	-1	0	-1	-1	-1	-1	-1	-1	-2

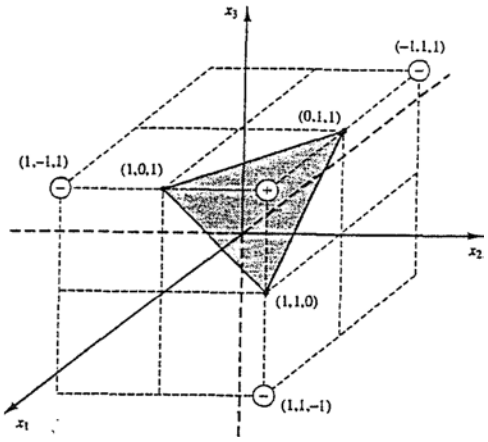
These resulting weights and bias still yield the wrong output (-1 = f(-1-1-1-2)) for the first pattern.

Hebbian Learning limitations for bipolar data:

We can show that even if both the patterns and the targets are encoded in bipolar format, the results of Hebbian learning may not be correct for all the training patterns:

Inputs					changes				weights			
x1	x2	x3	1	t	w1	w2	w3	b	w1	w2	w3	b
1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	-1	1	-1	-1	-1	1	-1	0	0	2	0
1	-1	1	1	-1	-1	1	-1	-1	-1	1	1	-1
-1	1	1	1	-1	1	-1	-1	-1	0	0	0	-2

We realize that even now the final weights do not yield the correct response for the first training pattern: (-1 = f(-2)). On the other hand we can see in the figure below that this problem is also linearly separable:



Note that at any point in Hebbian training the weights tend to satisfy the pattern - target combination that defined the last weight changes, but not necessarily pattern-target combinations used in *previous* learning steps.

In light or this, there are some points that we should note:

1) *The modification of the weights is entirely defined by the pattern-target combination at each step, disregarding the status that the network has reached up to that point.*

2) *This learning algorithm never checks to see how well the network is doing (the activation is never computed during training!)*

3) *It never checks how good the overall resulting weights are for classifying ALL the training patterns.*

4) *The modification of the weights from one pattern to the other seem to be to "radical", favoring the correct classification of the latest training pattern but possibly disrupting the correct classification of previous patterns.*

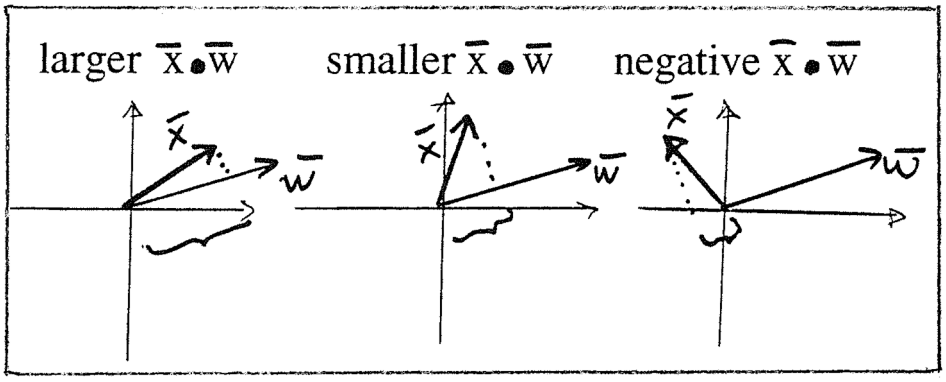
Calculation of the net input as a vector projection

From the vector expression for the net input:

$$y\_net = \Sigma\, x_i\, w_i = \bar{x} \cdot \bar{w}$$

We realize that y_net is the *inner product* of the input vector x and the weight vector w (note that these vectors may include entries for the "bias" and its associated "1")

We also recall that *the inner product represents a measure of alignment of two n-dimensional vectors*:



Then, in the processing element, an *input vector x will achieve a larger y_net if it is well aligned with the weight vector w*.

Learning as a redirection of the weight vector

We now can think of a 'learning step'(i.e., the modification of the weights by the presentation of a single pattern-target combination), as *a realignment of the weight vector so that it will generate an appropriate inner product for that pattern* (very positive if the target is +1 and very negative if the target is -1). After learning from all the training patterns, the final weight vector should be "directed" in such a way that it yields a large, positive inner product for all the patterns that had a +1 target. *Graphically, it means that after training the weight vector should "point" to the cluster of training patterns from the "ON" (t = +1) class.*

Consider the last exercise in Hebb learning from this point of view *(every pattern presentation causes the weight vector to realign toward or away from the input pattern, for t= +1 or t= -1, respectively)*: