Writing A for the activation,

$$A = \sum_{i=0}^{n} w_i I_i$$

where I_n is taken to be the bias input, 1. This may be written in vector notation

$$A = \mathbf{W}.\mathbf{I}$$

where **W** is the weight vector, and **I** is the input vector (extended by a 1 in the n+1'th position)

The output is

$$\begin{array}{l} 1 \text{ if } A \geq 0 \\ 0 \text{ if } A < 0 \end{array}$$

The weight change rule is:

$$\Delta w_i = \eta(D - Y)I_i$$

This rule is applied pattern by pattern:

that is, as each input/output pair is applied to the network, the weights are altered.

Notes:

If the input pattern produces the correct output pattern using the current weight vector, then no alteration in the weights will be made. (D-Y will be 0)

If the input pattern does *not* produce the correct output, the weight change made amounts to altering the weight vector by an amount proportional to the extended input vector:

if (D-Y) = 1

$$\Delta\mathbf{W} = \eta\mathbf{I}$$

or if (D-Y) = -1

$$\Delta\mathbf{W} = -\eta\mathbf{I}$$

The most usual way of applying this type of learning rule is to apply it *epoch by epoch, updating the weights after each pattern presentation,* that is,

```
Initialise W

repeat

    for each input pattern (j=1 to M)

        apply the input pattern Ij
        compute the output Y for this
                                pattern
        if (D ≠ Y)
            compute the ΔW from Y, D, and I
            alter the weights by ΔW

until no weights at all have been changed
       in any of the pattern presentations
```

The presentation of a complete set of patterns (and the alteration of the weights) is often known as an *epoch*.

Example:
Learning to be an AND gate.
Learning to be an OR gate.

The algorithm is extremely general. It can be shown that if a solution exists, then the algorithm will find it within a finite number of steps. See Fausett, pp 76-80 or Haykin 108-112 for proof of this.

Linear Separability.

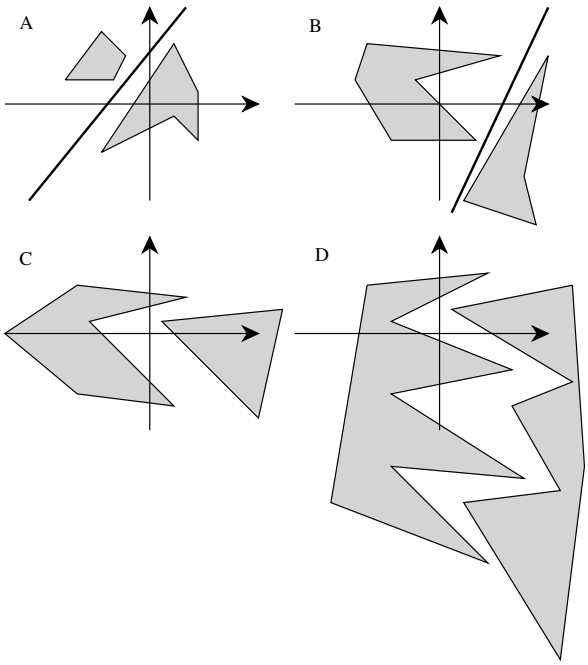
Since a perceptron will find the solution in weight-space if one exists, we become interested in what can and cannot be solved with a single perceptron.

Recall that the decision surface was a (hyper)plane of dimension N-1 for and N dimensional input space.

Two sets of points in N-space are said to be *linearly separable* if they can be separated by an N-1 dimensional hyperplane.

Thus we consider the inputs to some network as being points in N-space (and this is a reasonable geometric analogue for a system with N inputs). If the set of points for which we have a target output of 0 is linearly separable from the set of points for which we have a target output of 1, then the classification task is learnable by a simple perceptron.

If there are two inputs, this is easily visualisable. But it is not so easy if there are many inputs!



A and B are linearly separable: C and D are not linearly separable.