2: TLUs and vectors - simple learning rules

Kevin Gurney

Dept. Human Sciences, Brunel University
Uxbridge, Middx. UK

1 Geometric interpretation of TLU action

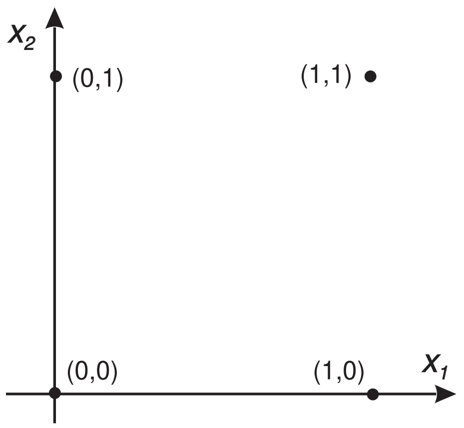
It is possible to interpret the functionality of a TLU geometrically. In summary, it separates its input space into two parts divided by a hyperplane according to whether the input is classified as a '1' or a '0'. We now introduce the ideas contained here step by step.

1.1 Pattern classification and input space

Consider the TLU with weights $w_1 = 1, w_2 = 1$ and threshold 1.5. The table of responses is shown below.

x_1	x_2	activation	output
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

The TLU may be thought of as *classifying* its input patterns into two classes: those that give output '1' and those that give output '0'. Each input pattern has two *components*, x_1, x_2. We may therefore represent these patterns in two-dimensional space



2-d pattern space

The space in which the inputs reside is referred to as the *pattern space*. Each pattern determines a point in the space by using its component values as space-coordinates. In general, for n-inputs, the pattern space will be n-dimensional. Clearly, for $n > 3$ the pattern space cannot be drawn or represented in physical space. This is not a problem: we shall return to the idea of using higher dimensional spaces later. However, the geometric insight obtained in 2-D will carry over (when expressed algebraically) into n-D.

1.2 The linear separation of classes

Since the critical condition for classification occurs when the activation equals the threshold, it is useful to examine the geometric implication of this. Putting the activation equal to the threshold gives

$$\sum_{i=1}^{n} w_i x_i = \theta \tag{1}$$

In the 2-D case we are considering

$$w_1 x_1 + w_2 x_2 = \theta \tag{2}$$

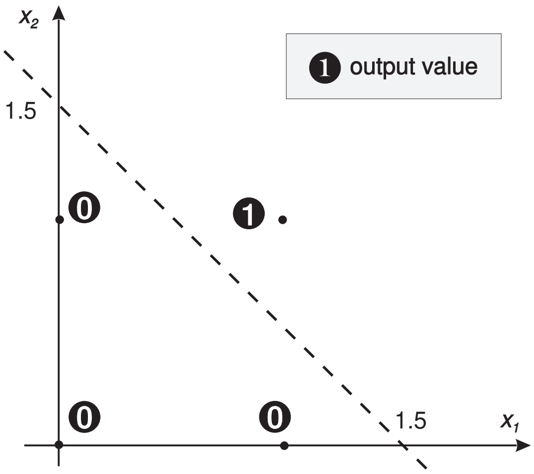
so that

$$x_2 = -\left(\frac{w_1}{w_2}\right) x_1 + \left(\frac{\theta}{w_2}\right) \tag{3}$$

This is of the form

$$x_2 = a x_1 + b \tag{4}$$

That is, a straight line with slope a and intercept b on the x_2 axis. Since $w_1 = w_2 = 1$ and $a = -1$, we also have $b = 1.5$.

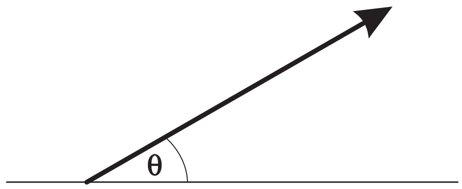


Line in 2-d pattern space

The two classes are therefore separated by the 'decision' line which is defined by putting the activation equal to the threshold. It turns out that it is possible to generalise this result to TLUs with n inputs In 3-D the two classes are separated by a *decision-plane*. In n-D this becomes a *decision-hyperplane*. (The 'hyper-' is sometimes dropped even when $n > 3$). The converse of this is that, any TLU is defined by some hyperplane in its pattern space and any function which cannot be realised in this way cannot be realised by a TLU. Because the defining equation (1) for the hyperplane, is *linear*, the TLU is a *linear classifier*. Using some ideas about *vectors*, it is possible to prove these results and to gain insight into what is going on.

2 Vectors

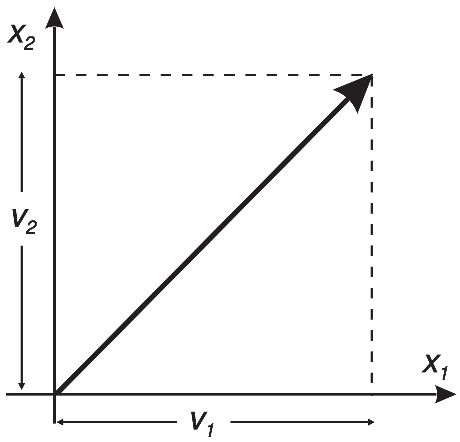
Vectors* are usually introduced as representations of quantities that have magnitude and direction. Thus the velocity of a car is defined by the car's speed and direction. On paper we may draw an arrow whose length is proportional to the speed and whose direction is the same as that of the car.



2-D vector

Vectors will be denoted by bold face letters e.g. $\mathbf{v}$. The magnitude of $\mathbf{v}$ will be denoted by $\|\mathbf{v}\|$. In writing vectors we can't use bold so we usually put an underline thus – $\underline{v}$. The length of vectors is sometimes denoted by the italic letter e.g. v. In accordance with our geometric ideas a vector is now defined by the pair of numbers $(\|\mathbf{v}\|, \theta)$ where θ is the angle the vector makes with some reference direction.

In order to generalise to higher dimensions, and to relate vectors to the ideas of pattern space, it is more convenient to describe vectors with respect to a cartesian coordinate system. That is, we give the projected lengths of the vector onto two perpendicular axes



vector wrt cartesian axes

The vector is now described by the pair of numbers v_1, v_2. These numbers are its *components* in the chosen coordinate system. Since they completely determine the vector we may think of the vector itself as a pair of component values and write $\mathbf{v} = (v_1, v_2)$. The

*For a good introduction to vectors as required for neural nets, see chapter 9 in PDP vol. 1

vector is now an ordered list of numbers. Notice the ordering is important, since (1,3) is in a different direction from (3,1).

This definition immediately generalises to more than 2-dimensions. An n-dimensional vector is simply an ordered list of n numbers, $\mathbf{v} = (v_1, v_2, \dots v_n)$. Lists like this appeared on the first problem sheet. They were the *weight vector* $(w_1, w_2, \dots, w_n)$ and the *input vector* $(x_1, x_2, \dots x_n)$ for the node.

2.1 The length of a vector

For our 2-D prototype, the length of a vector is just its length in the plane. In terms of its components, this is given by pythagoras's theorem.

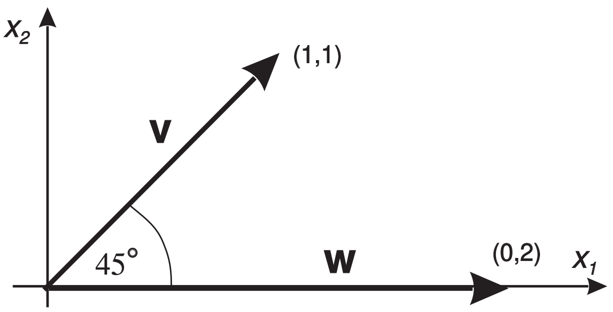
$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2} \tag{5}$$

In n-dimensions, the length is *defined* by the natural extension of (5)

$$\|\mathbf{v}\| = \left[\sum_{i=1}^{n} v_i^2 \right]^{\frac{1}{2}} \tag{6}$$

2.2 Comparing vectors - the inner product

Consider the vectors $\mathbf{v} = (1,1)$ and $\mathbf{w} = (0,2)$ shown below



Two vectors at $45°$

The angle between them is $45°$. Define the *inner product* $\mathbf{v} \cdot \mathbf{w}$ of the two vectors by

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 \tag{7}$$

The form on the right-hand side should be familiar - it is just the same as that we have used to define the activation of a TLU... Substituting the component values gives $\mathbf{v} \cdot \mathbf{w} = 2$. We will now try and give this a geometric interpretation.

First we note that $\|\mathbf{v}\| = \sqrt{2}$ and $\|\mathbf{w}\| = 2$. Next we observe that the *cosine* of the angle between the vectors is $1/\sqrt{2}$

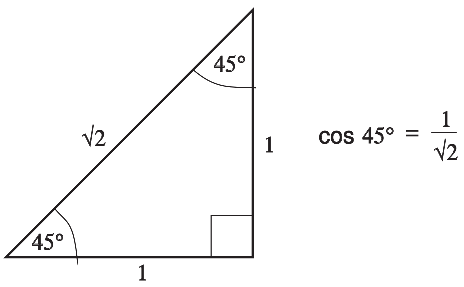


diagram of cos 45

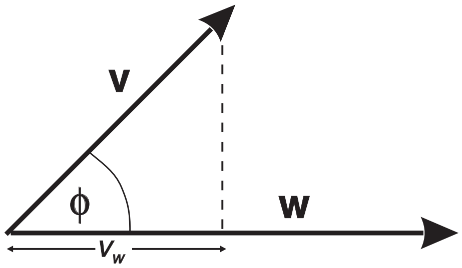
Therefore $\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\| \|\mathbf{w}\| \cos 45°$. It may be shown that this is a general result; that is, if the angle between two vectors $\mathbf{v}$ and $\mathbf{w}$ is ϕ then the definition of dot-product given in (7) is equivalent to

$$\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\| \|\mathbf{w}\| \cos \phi \tag{8}$$

(For a proof of this see ch 9 PDP vol 1). In n-dimensions the inner product is defined by the natural extension of (7)

$$\mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{n} w_i v_i \tag{9}$$

Although we cannot now draw the vectors (for $n > 3$) the geometric insight we obtained for 2-D may be carried over since the behaviour must be the same (the definition is the same). We therefore now examine the significance of the inner product in 2 dimensions. Essentially the inner product tells us how well 'aligned' two vectors are. To see this, let v_w be the component of $\mathbf{v}$ along the direction of $\mathbf{w}$, or the *projection* of $\mathbf{v}$ along $\mathbf{w}$.



Vector projection

The projection is given by $\|\mathbf{v}\| \cos \phi$ which, by (8) gives us

$$v_w = \frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{w}\|} \tag{10}$$

If ϕ is small then $\cos \phi$ is close to its maximal value of one. The inner product, and hence the projection v_w, will be close to its maximal value; the vectors 'line up' quite well. If $\phi = 90°$, the cosine term is zero and so too is the inner product. The projection is zero because the vectors are at right angles; they are said to be *orthogonal.* If $90 < \phi < 270$, the cosine is negative and so too, therefore, is the projection; its magnitude may be large but the negative sign indicates that the two vectors point into opposite half-planes.

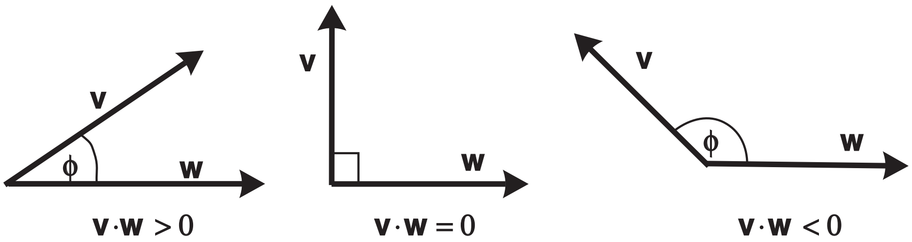


diagram of inner product cases

2.3 Inner products and TLUs

Using the ideas developed above we may express the action of a TLU in terms of the weight and input vectors. The activation a may now be expressed as

$$a = \mathbf{w} \cdot \mathbf{x} \tag{11}$$

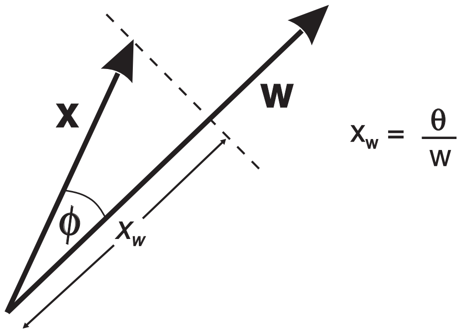
The vector equivalent to (1) now becomes

$$\mathbf{w} \cdot \mathbf{x} = \theta \tag{12}$$

If $\mathbf{w}$ and θ are constant then this implies the projection x_w, of $\mathbf{x}$ along $\mathbf{w}$ is constant since

$$x_w = \frac{\theta}{\|\mathbf{w}\|} \tag{13}$$

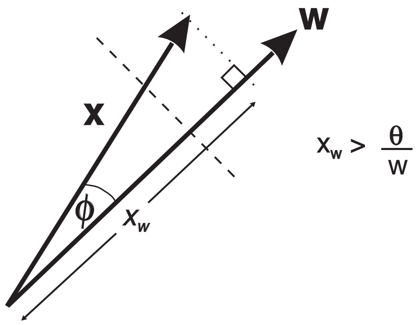
In 2-D, therefore, the equation specifies all $\mathbf{x}$ which lie on a line perpendicular to $\mathbf{w}$.



w.x = constant

There are now two cases

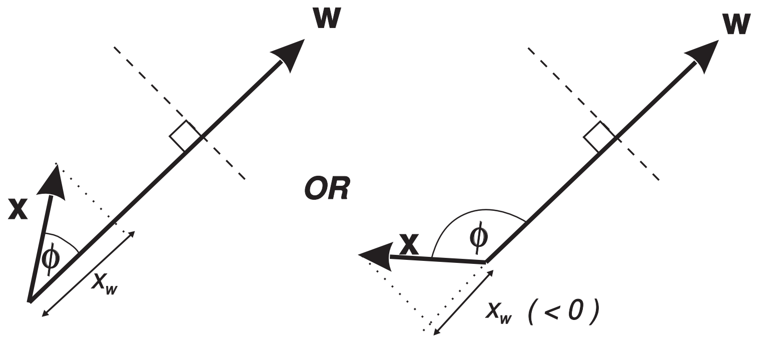
1. If $x_w > \theta/\|\mathbf{w}\|$, then $\mathbf{x}$ must lie beyond the dotted line



$\mathbf{w.x} > \theta$

Using (10) we have that $x_w > \theta/\|\mathbf{w}\|$ implies $\mathbf{w} \cdot \mathbf{x} > \theta$; that is, the activation is greater than the threshold

2. Conversely, if $x_w < \theta$ then $\mathbf{x}$ must lie *below* the dotted line



$$x_w < \frac{\theta}{w}$$

$$\mathbf{w.x} < \theta$$

Using (10) we have that $x_w < \theta/\|\mathbf{w}\|$ implies $\mathbf{w}\cdot\mathbf{x} < \theta$; that is, the activation is less than the threshold

In 3-D the line becomes a plane intersecting the cube and in n-D a hyperplane intersecting the n-dimensional hypercube or n-cube.

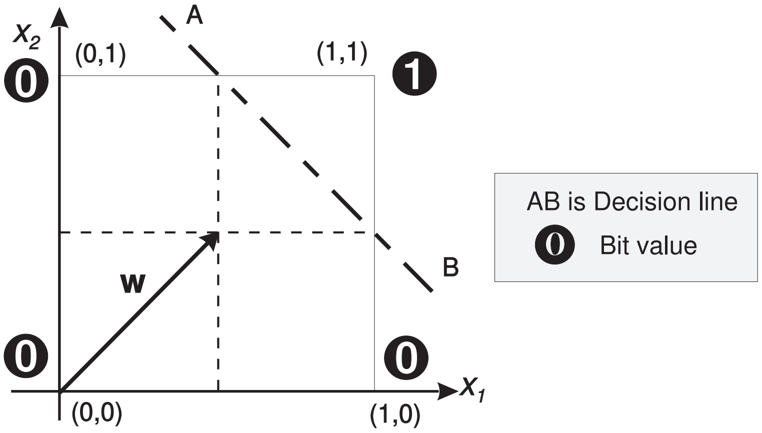
3 Training TLUs

Training any unit consists of adjusting the weight vector and threshold so that the desired classification is performed. In order to place the adaptation of the threshold on the same footing as the weights, we suppose that it is another weight which is permanently connected to an input of -1. I shall call this the *augmented* weight vector, in contexts where confusion might arise, although this terminology is by no means standard. Then for all TLUs we may express the node function as follows

$$\begin{array}{lcl} \mathbf{w}\cdot\mathbf{x} \geq 0 & \Rightarrow & y = 1 \\ \mathbf{w}\cdot\mathbf{x} < 0 & \Rightarrow & y = 0 \end{array} \tag{14}$$

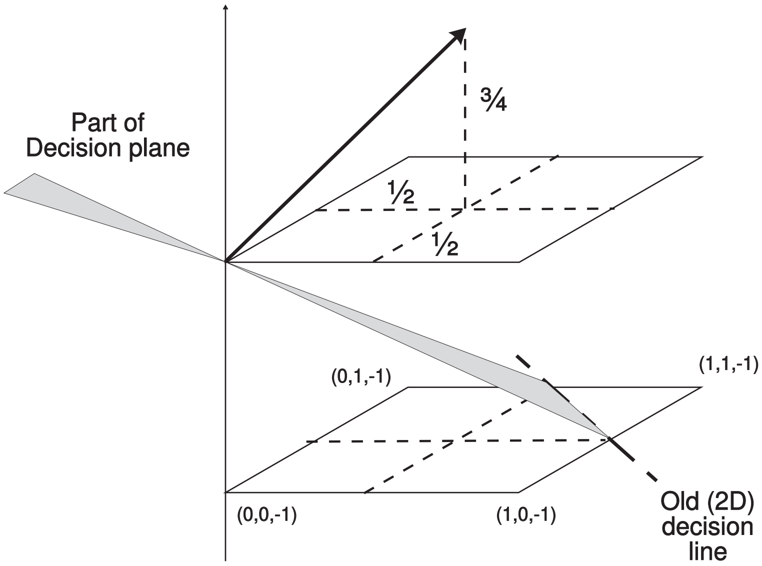
Putting $\mathbf{w}\cdot\mathbf{x} = 0$ now defines the decision (hyper)plane. This plane is therefore orthogonal to the (augmented) weight vector and passes through the origin of the (augmented) pattern space.

To see this consider the example below



AND gate with w = (1/2, 1/2, 3/4)

The result in the augmented space is shown below

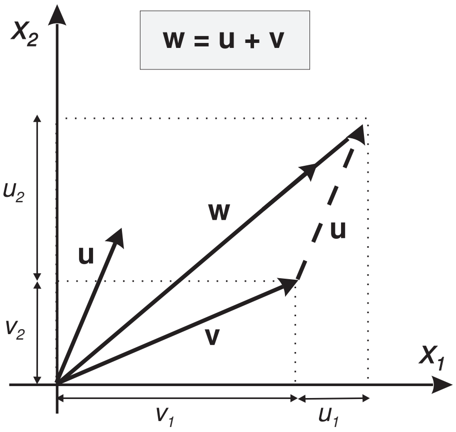


Augmented pattern space in 3D

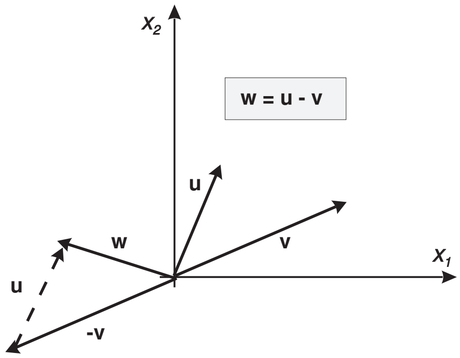
Changing any vector (and hence the weight vector in particular) may be thought of as adding another vector to it. Two vectors $\mathbf{u} = (u_1, u_2)$, $\mathbf{v} = (v_1, v_2)$ may be added by summing their components to give a new vector $\mathbf{w} = (w_1, w_2)$ where

$$\begin{aligned} w_1 &= u_1 + v_1 \\ w_2 &= u_2 + v2 \end{aligned} \tag{15}$$

Subtraction may be defined by noting that the negative of a vector is just the same vector pointing in the opposite direction. Then subtraction of $\mathbf{v}$ is just addition of $-\mathbf{v}$. These operations are shown below.



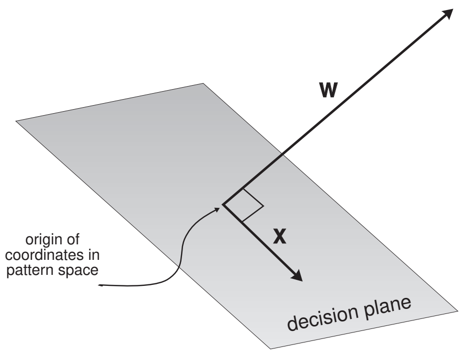
Vector addition



Subtraction of a vector

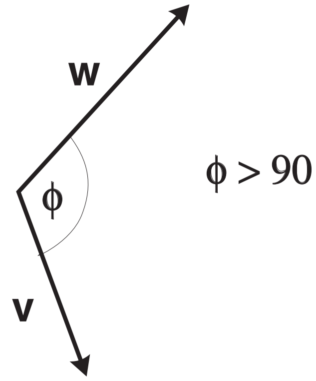
3.1 Adjusting the weight vector

We require the weight vector to be orthogonal to the decision plane and that the plane must pass through the origin.



Orthogonal/origin requirement

The training set for the TLU will consist of a set of pairs $\{\mathbf{v}, t\}$, where $\mathbf{v}$ is an input vector and t is the target class or output ('1' or '0') that $\mathbf{v}$ belongs to. This type of training is known as *supervised* training, since we tell the net what output is expected for each vector. Suppose we present a training vector $\mathbf{v}$ to the TLU whose current weight vector is $\mathbf{w}$. Further, suppose that $t = 1$ but that, with the weight vector as it is, it produces an output of $y = 0$. To produce a '0' the activation must have been negative when it should have been positive. The situation is shown below.

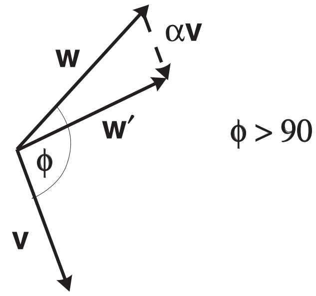


Misclassification 1 - 0

In order to correct the situation we need to rotate $\mathbf{w}$ so that it points more in the direction of $\mathbf{v}$. At the same time, we don't want to make too drastic a change as this might upset previous learning. We can achieve both goals by adding a fraction of $\mathbf{v}$ to $\mathbf{w}$ to produce a new weight vector $\mathbf{w}'$; that is

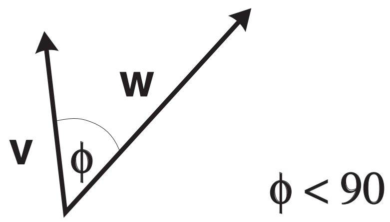
$$\mathbf{w}' = \mathbf{w} + \alpha\mathbf{v} \tag{16}$$

where $0 < \alpha < 1$



increment made to weight vector in case 1

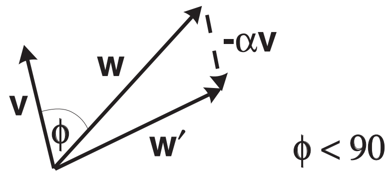
Suppose now instead, that $t = 0$ but $y = 1$. This means the activation was positive when it should have been negative.



Misclassification 0 - 1

We now need to rotate $\mathbf{w}$ *away* from $\mathbf{v}$ which may be effected by *subtracting* a fraction of $\mathbf{v}$ from $\mathbf{w}$; that is

$$\mathbf{w}' = \mathbf{w} - \alpha\mathbf{v} \tag{17}$$



Increment made to weight vector in case 2

Both (16) and (17) may be written as a single rule as follows

$$\mathbf{w}' = \mathbf{w} + \alpha(t - y)\mathbf{v} \tag{18}$$

This may be written in terms of the change in the weight vector[†] $\Delta\mathbf{w}$ or in terms of the components

$$\Delta w_i = \alpha(t - y)v_i \tag{19}$$

This is our first example of a *training rule* or *learning rule*. The parameter α is called the *learning rate*. The learning rule may be incorporated into a *training algorithm* for TLUs as follows

repeat
    for each training vector pair $(\mathbf{v}, t)$
        evaluate the output y when $\mathbf{v}$ is input to the TLU
        if $y \neq t$ then
            form a new weight vector $\mathbf{w}'$ according to (18)
        else
            do nothing
        end if
    end for
until $y = t$ for all vectors

This is usually known as the *Perceptron learning algorithm* because it was used extensively with an extension of the TLU known as the perceptron described in the next section. We now look at an example of using this method with an ordinary TLU.

3.2 Example

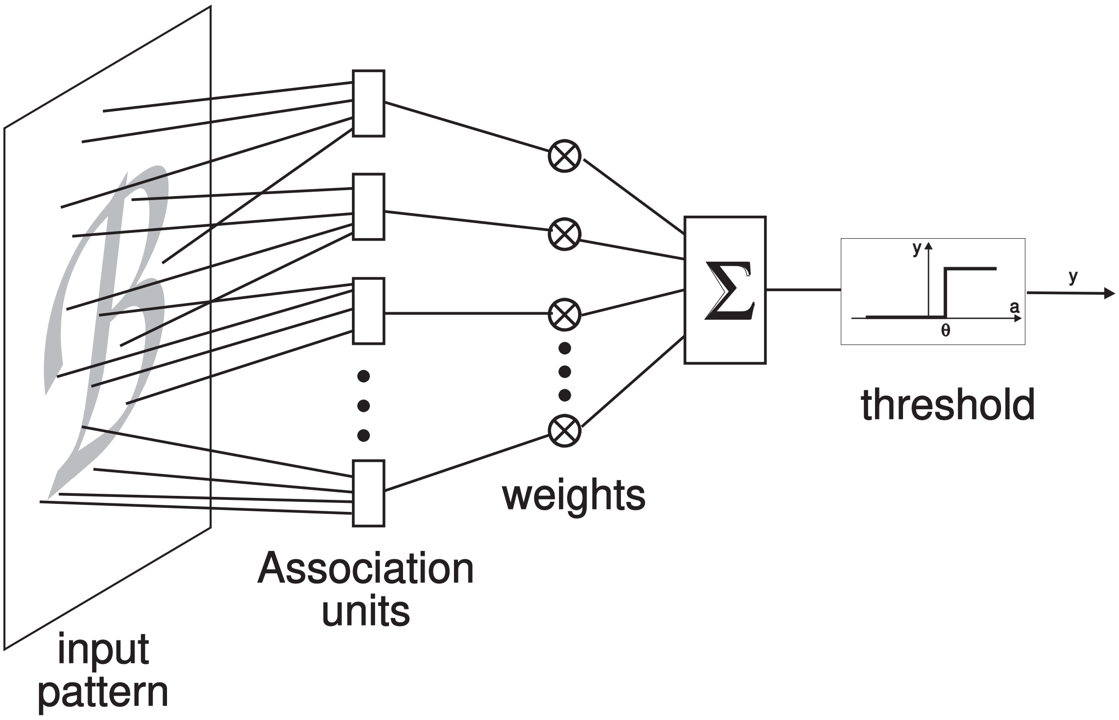
A TLU has weights 0, 0.4 and threshold 0.3. It has to learn the logical AND function; [all outputs '0' except with input (1,1)]. The learning rate is 0.25. Using the above algorithm, the following sequence of events takes place.

4 The Perceptron

This is an enhancement of the TLU introduced by Rosenblatt (Rosenblatt, 1962). It consists of a TLU whose inputs come from a set of preprocessing *association units* (A-units).

[†]In general, a change in a quantity is denoted by the by the Greek letter 'delta'; either upper case Δ or lower case δ

w_1	w_2	θ	x_1	x_2	a	y	t	$\alpha(t-y)$	δw_1	δw_2	$\delta\theta$
0.0	0.4	0.3	0	0	0	0	0	0	0	0	0
0.0	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	0.25
0.0	0.15	0.55	1	0	0	0	0	0	0	0	0
0.0	0.15	0.55	1	1	0.15	0	1	0.25	0.25	0.25	-0.25
0.25	0.4	0.3	0	0	0	0	0	0	0	0	0
0.25	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	0.25
0.25	0.15	0.55	1	0	0.25	0	0	0	0	0	0
0.25	0.15	0.55	1	1	0.4	0	1	0.25	0.25	0.25	-0.25
0.5	0.4	0.3	0	0	0	0	0	0	0	0	0
0.5	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	0.25
0.5	0.15	0.55	1	0	0.5	0	0	0	0	0	0



Perceptron

The A-units can be assigned any arbitrary Boolean functionality but are fixed - they do not learn. The rest of the node functions just like a TLU and may be therefore be trained in exactly the same way. Rosenblatt was the first to use the training algorithm described here - hence the name ‘perceptron training’.

References

Rosenblatt, F. (1962). *Principles of Neurodynamics.* Spartan Books.