3: The delta rule

Kevin Gurney

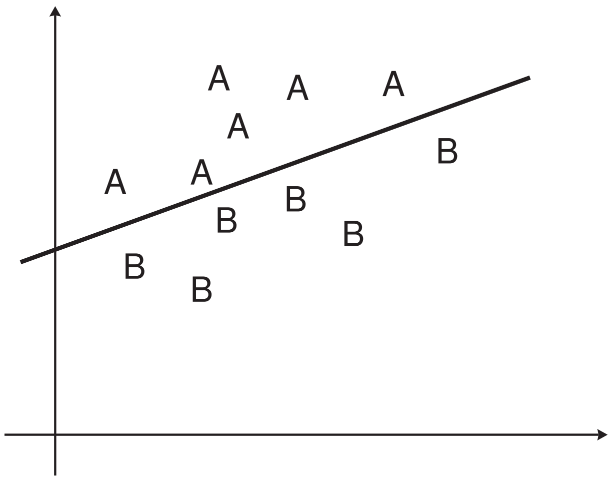
Dept. Human Sciences, Brunel University
Uxbridge, Middx. UK

1 Generating input vectors for Neural Nets

In order to make the potential applications discussed subsequently more concrete, we shall consider the example of how image information may be captured and input to a network. Suppose we have a TV camera (monochrome for simplicity) which is viewing a picture that is to be used in training. The output from this is a picture where each point is represented by a continuously variable voltage (analogue quantity) so that shades of grey may be encoded accurately. For a perceptron, however, we require a set of Binary values ('1', '0'). The conversion process is done by dividing the picture into a grid of picture elements or *pixels* each of which is allowed to take only one of two values - black or white. To find the value for each pixel, the average value of the image in the pixel area is found and then thresholded to determine whether it is white or black. We now make the correspondence white = '1', say and black = '0'. This array of Boolean quantities may now be stored in a special purpose computer memory or *framestore*. Typically the pixel grid may be 512 by 512 giving over 1/4 million pixels. Thus, the pattern space will have dimension 1/4 million. This is often reduced to make things more manageable.

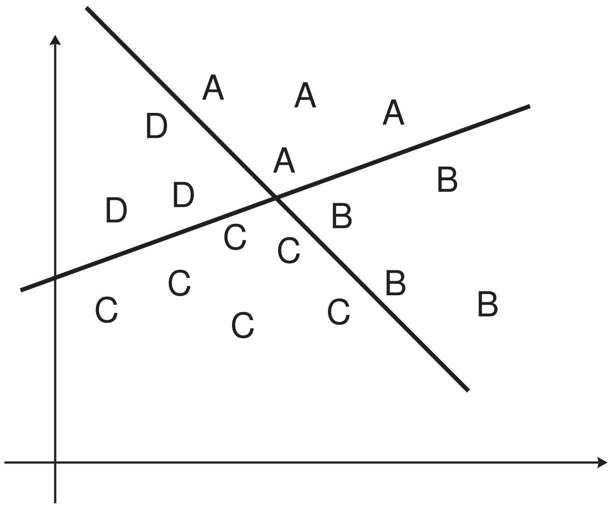
2 Using TLUs and perceptrons as classifiers

Using the perceptron training algorithm, we may now use a perceptron to classify two linearly separable classes A and B. Examples from these classes may have been obtained, for example, by capturing images in a framestore; there may be two classes of faces, or we want to separate handwritten characters into numerals and letters.



A/B classification

Suppose now there are 4 classes *A, B, C, D* and that they are separable by two planes in pattern space



pattern space for A B C D

That is the two classes *(A,B) (C,D)* are linearly separable, as too are the classes *(A,D)* and *(B,C)*.

We may now train two units (with outputs y_1, y_2) to perform these two classifications

	1	0
y_1	(A B)	(C D)
y_2	(A D)	(B C)

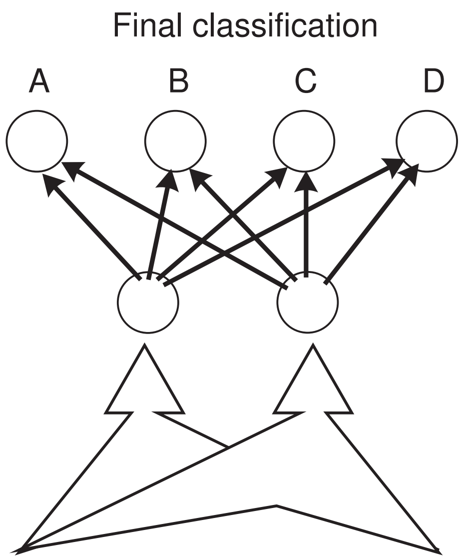
y1 y2 classification

This gives a table encoding the original 4 classes

y_1	y_2	Class
0	0	C
0	1	D
1	0	B
1	1	A

y1 y2 coding for A B C D

The output of the two units may now be decoded by four 2-input TLUs to give the desired responses



2 layer net giving A B C D classification

These *output units* are not trained; each one is assigned weights required to signal a '1' when its class code appears at its inputs. For example, output unit 'A' is the logic AND gate given as an example at the beginning of lecture 2.

Notice that the grouping (A,C) (D,B) would not have worked, since these are not linearly separable, and other arrangements of the four classes in pattern space will require a different set of groupings. There were therefore two pieces of information required in order to train the two units.

1. The four classes may separated by 2-hyperplanes

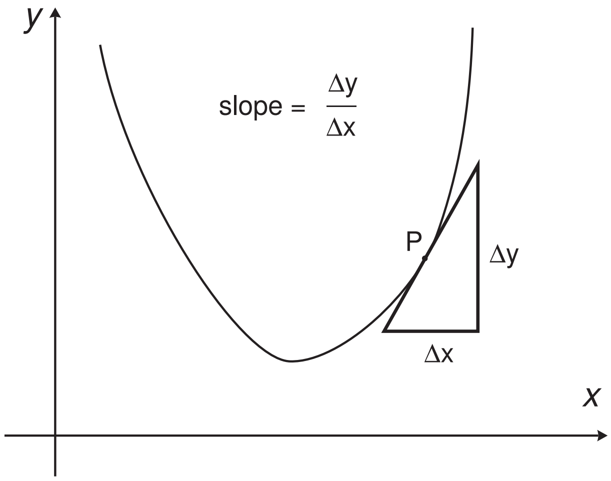
2. *(A,B)* was linearly separable from *(C,D)* and *(A,D)* was linearly separable from *(B,C)*.

It would be more satisfactory if we could dispense with 2) and train the entire 2-layer architecture, shown above, as a whole *ab initio*. The less knowledge we have to glean by ourselves, the more useful a network is going to be. In order to do this, it is necessary to introduce a new training algorithm based on a slightly different approach which obviates the need to know the nature of the nodes' hyperplanes.

3 Minimising an error: the delta rule

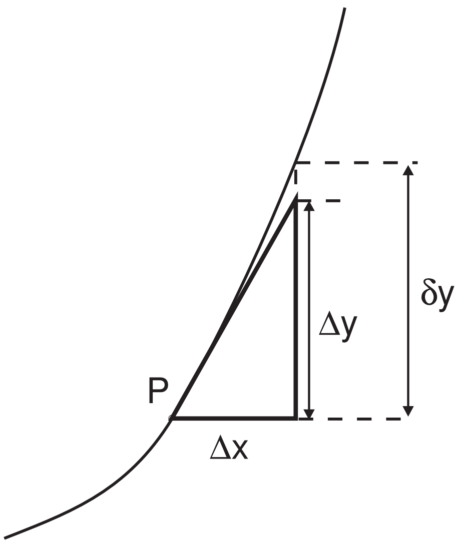
3.1 Finding the minimum of a function: gradient descent

Suppose y is some function of x (y depends on x or $y = y(x)$) but we don't know the exact form of this function. Further, suppose we wish to find the position (x-coordinate) of the minimum value of the function and we can find the slope (rate of change of y) at any point. The slope is just $\Delta y/\Delta x$ in the diagram.



y = y(x) and slope

The slope of a function at any point is the gradient (cf hill gradients) of the tangent to the curve at the point. If Δx is small, then Δy is almost the same as the change δy in the function y, when the change Δx is made in x.



small changes

That is

$$\delta y \approx \Delta y = \frac{\Delta y}{\Delta x}\Delta x \tag{1}$$

so that

$$\delta y \approx \text{slope} \times \Delta x \tag{2}$$

Now put

$$\Delta x = -\alpha \times \text{slope} \tag{3}$$

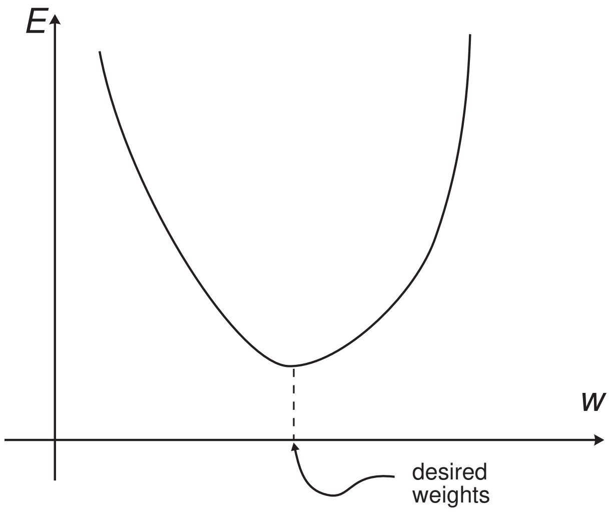
where $\alpha > 0$ and is small enough to ensure $\delta y \approx \Delta y$ then

$$\delta y \approx -\alpha(\text{slope})^2 \tag{4}$$

That is $\delta y < 0$ and we have 'travelled down' the curve towards the minimal point. If we keep repeating steps like (4) iteratively, then we should approach the value of x associated with the function minimum. This technique is called *gradient descent*. How can this be used to train networks?

3.2 gradient descent on an error

The idea is to calculate an error each time the net is presented with a training vector (given that we have supervised learning where there is a target) and to perform a gradient descent on the error considered as function of the weights. There will be a gradient or slope for each weight. Thus, we find the weights which give the minimal error. The situation is as follows.



gradient descent - E vs w

Formally, for each pattern p, we assign an error E_p which is a function of the weights; that is $_p = E_p(w_1, w_2, \ldots, w_n)$. Typically this is defined by the square difference between the output and the target. Thus (for a single node)

$$E_p = \frac{1}{2}(t-y)^2 \tag{5}$$

Where we regard y as a function of the weights. The total error E, is then just the sum of the pattern errors

$$E = \sum_p E_p \tag{6}$$

Now, in order to perform gradient descent, the error must be a continuous function of the weights and there must be a well defined gradient at each point. With TLUs, however, this is not the case; although the *activation* is a continuous function of the weights, the output changes abruptly as the activation passes throught the threshold value.

One way to remedy this is to train on the activation itself rather than the output. This technique is usually ascribed to Widrow and Hoff (Widrow and Hoff, 1960) who trained TLUs which had had their outputs labelled -1, 1 instead of 0, 1. These units they called Adaptive Linear Elements or ADALINEs. For a description of their techniques see (Widrow and Stearns, 1985; Widrow et al., 1987). The learning rule based on gradient decsent with this type of node is, therefore, sometimes known as the Widrow Hoff rule, but more usually now, as the *delta rule*.

We must still supply a target which is the activation the node is supposed to give in response to the training pattern. Recall (lecture 2) that, if the threshold of a TLU is treated as a weight, the condition for classifying as a '1' was that the activation, should be greater (or equal to) zero. Conversely for a '0' to be output we require the activation to be less than zero. We may therefore choose, as our target activations for the two classes, any two numbers of opposite sign. It is convenient to choose the set $\{-1, 1\}$.

The learning rule may now be obtained by finding the slope of the error in (5) with respect to ('wrt') each of the weights, but using activation a rather than output y. That is, for the delta rule with TLUS

$$E_p = \frac{1}{2}(t-a)^2 \tag{7}$$

It may be shown (use of 'function-of-a-function' in calculus) that the slope of E_p with respect to w_j is just $-(t-a)x_j$. The learning rule (delta rule) is now defined by making a change in the weight Δw_j in line with (3)

$$\begin{aligned} \Delta w_j &= -\alpha \times (\text{slope of } E_p \text{ wrt } w_j) \\ &= \alpha(t-a)x_j \end{aligned} \tag{8}$$

This rule may incorporated into a training algorithm similar to the one given in lecture 2. However, the error will never be exactly zero and so the possibility of 'do nothing' given there, will never arise with the delta rule - there will always be some update to the weights. The term $\alpha(t-a)$ is sometimes know as the 'delta' (or δ).

An example of this rule is provided below in which we train the same TLU as used in the Perceptron example of lecture 2 [initial weights (0, 0.4) threshold 0.3, learn rate 0.25].

v	w_1	w_2	θ	x_1	x_2	$a-\theta$	t	δ	δw_1	δw_2	$\delta\theta$	E
1	0.00	0.40	0.30	0	0	-0.30	-1.00	-0.17	-0.00	-0.00	0.17	0.24
2	0.00	0.40	0.48	0	1	-0.08	-1.00	-0.23	-0.00	-0.23	0.23	0.43
3	0.00	0.17	0.71	1	0	-0.71	-1.00	-0.07	-0.07	-0.00	0.07	0.04
4	-0.07	0.17	0.78	1	1	-0.68	1.00	0.42	0.42	0.42	-0.42	1.42
1	0.35	0.59	0.36	0	0	-0.36	-1.00	-0.16	-0.00	-0.00	0.16	0.21
2	0.35	0.59	0.52	0	1	0.07	-1.00	-0.27	-0.00	-0.27	0.27	0.57
3	0.35	0.32	0.79	1	0	-0.44	-1.00	-0.14	-0.14	-0.00	0.14	0.16
4	0.21	0.32	0.93	1	1	-0.40	1.00	0.35	0.35	0.35	-0.35	0.98
1	0.56	0.67	0.58	0	0	-0.58	-1.00	-0.11	-0.00	-0.00	0.11	0.09
2	0.56	0.67	0.68	0	1	-0.01	-1.00	-0.25	-0.00	-0.25	0.25	0.49
3	0.56	0.42	0.93	1	0	-0.37	-1.00	-0.16	-0.16	-0.00	0.16	0.20
4	0.40	0.42	1.09	1	1	-0.26	1.00	0.32	0.32	0.32	-0.32	0.80
1	0.72	0.74	0.77	0	0	-0.77	-1.00	-0.06	-0.00	-0.00	0.06	0.03
2	0.72	0.74	0.83	0	1	-0.09	-1.00	-0.23	-0.00	-0.23	0.23	0.42
3	0.72	0.51	1.06	1	0	-0.34	-1.00	-0.16	-0.16	-0.00	0.16	0.22
4	0.55	0.51	1.22	1	1	-0.16	1.00	0.29	0.29	0.29	-0.29	0.67
1	0.84	0.80	0.93	0	0	-0.93	-1.00	-0.02	-0.00	-0.00	0.02	0.00
2	0.84	0.80	0.95	0	1	-0.15	-1.00	-0.21	-0.00	-0.21	0.21	0.36
3	0.84	0.59	1.16	1	0	-0.32	-1.00	-0.17	-0.17	-0.00	0.17	0.23
4	0.67	0.59	1.33	1	1	-0.07	1.00	0.27	0.27	0.27	-0.27	0.57
1	0.94	0.86	1.06	0	0	-1.06	-1.00	0.02	0.00	0.00	-0.02	0.00
2	0.94	0.86	1.05	0	1	-0.19	-1.00	-0.20	-0.00	-0.20	0.20	0.33
3	0.94	0.65	1.25	1	0	-0.31	-1.00	-0.17	-0.17	-0.00	0.17	0.24
4	0.77	0.65	1.42	1	1	-0.00	1.00	0.25	0.25	0.25	-0.25	0.50
1	1.02	0.90	1.17	0	0	-1.17	-1.00	0.04	0.00	0.00	-0.04	0.01
2	1.02	0.90	1.13	0	1	-0.22	-1.00	-0.19	-0.00	-0.19	0.19	0.30
3	1.02	0.71	1.32	1	0	-0.31	-1.00	-0.17	-0.17	-0.00	0.17	0.24
4	0.84	0.71	1.50	1	1	0.06	1.00	0.24	0.24	0.24	-0.24	0.44

First ‘correct pass’ through the training set. The following training decreases the error but does not change the classification after thresholding the activationAfter this the

v	w_1	w_2	θ	x_1	x_2	$a-\theta$	t	δ	δw_1	δw_2	$\delta\theta$	E
1	1.08	0.95	1.26	0	0	-1.26	-1.00	0.07	0.00	0.00	-0.07	0.03
2	1.08	0.95	1.20	0	1	-0.25	-1.00	-0.19	-0.00	-0.19	0.19	0.28
3	1.08	0.76	1.38	1	0	-0.30	-1.00	-0.17	-0.17	-0.00	0.17	0.24
4	0.91	0.76	1.56	1	1	0.11	1.00	0.22	0.22	0.22	-0.22	0.40
1	1.13	0.98	1.33	0	0	-1.33	-1.00	0.08	0.00	0.00	-0.08	0.06
2	1.13	0.98	1.25	0	1	-0.27	-1.00	-0.18	-0.00	-0.18	0.18	0.27
3	1.13	0.80	1.43	1	0	-0.30	-1.00	-0.17	-0.17	-0.00	0.17	0.24
4	0.95	0.80	1.61	1	1	0.15	1.00	0.21	0.21	0.21	-0.21	0.36

Examination of (8) shows that it looks formally the same as the perceptron rule [lecture 2]. However, the latter uses the output for comparison with a target, while the delta rule uses the activation. They were also obtained from different theoretical starting points. The perceptron rule was derived by a consideration of hyperplane manipulation while the delta rule is given by gradient descent on the square error.

It was noted above that the discontinuity in error for TLUs could be traced to the discontinuous output function. With semilinear units this is not the case since the sigmoid is a smooth function. Now we may use the error in (5) (using the output rather than the activation) but have to include an extra term which is related to the slope of the sigmoid; that is, the derivative $\sigma'(a)$. So for semilinear units the delta rule becomes

$$\Delta w_j = \alpha\sigma'(a)(t - y)x_j \tag{9}$$

It may be shown that

$$\sigma'(a) \equiv \frac{d\sigma(a)}{da} = \frac{1}{\sigma}\sigma(a)(1 - \sigma(a)) \tag{10}$$

Unlike the perceptron rule, it is possible to generalise the delta rule to train more than a single layer at once. It turns out to be possible to calculate the slope of the error gradient at intermediate network layers. This was our original goal and is fulfilled in the so-called Backpropagation algorithm or generalised delta rule to be dealt with in the next lecture.

References

Widrow, B. and Hoff (1960). Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pages 96 – 104. IRE.
Reprinted in *Neurocomputing - Foundations of Research* eds. Anderson and Rosenfeld. This is a third party report on Widrow's paper. It is largely of historic interest only.

Widrow, B. and Stearns, S. (1985). *Adaptive Signal Processing.* Prentice-Hall.
Is in the library short loan section. This is a book on signal processing (Widrow is an engineer) but contains an extensive analysis of gradient descent. The ADALINE stuff is in the first half of the book.

Widrow, B., Winter, and Baxter (1987). Learning phenomena in layered neural networks. In *1st Int. Conference Neural Nets, San Diego*, volume 2, page 411. I have this.This gives a nice description of training linear units and the ideas of linear separability.