REMOTE SENSING

MODULE OF

REMOTE SENSING DATA ANALYSIS

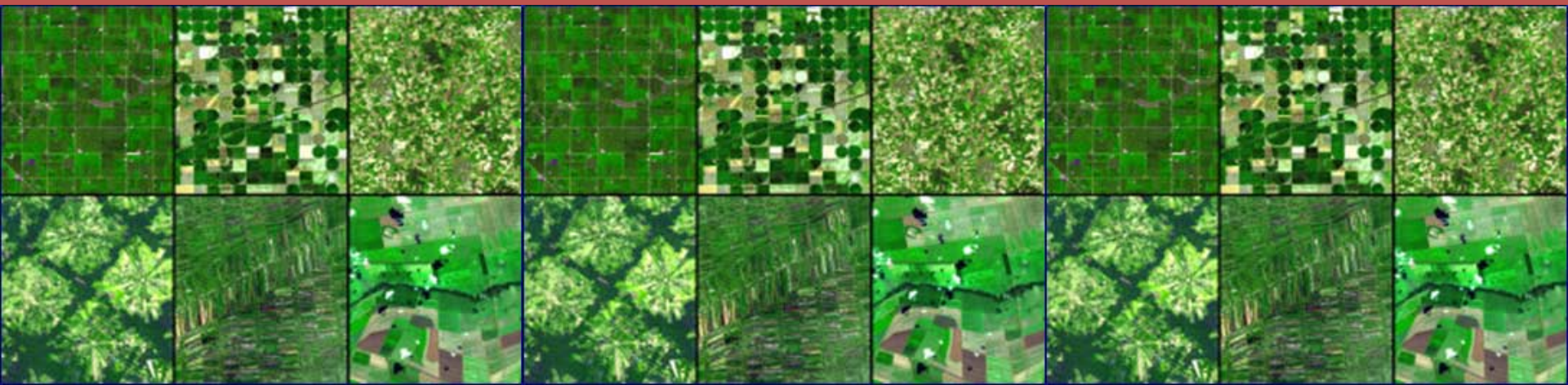
(6 CFU)

A.Y. 2013/14

MASTER OF SCIENCE IN COMMUNICATION TECHNOLOGIES AND MULTIMEDIA

PROF. ALBERTO SIGNORONI

SUPERVISED NON-PARAMETRIC CLASSIFICATION: GEOMETRIC APPROACHES



Introduction

- Statistical classification algorithms are the most commonly encountered labelling techniques used in remote sensing and, for this reason, have been the principal methods in our treatment.
  - One of the valuable aspects of a statistical approach is that a *set* of relative likelihoods is produced.
  - Even though, in the majority of cases, the *maximum* of the likelihoods is chosen to indicate the most probable label for a pixel, *there remains nevertheless information in the remaining likelihoods that could be made use of in some circumstances*,
    - either to initiate a process such as relaxation labelling
    - or simply to provide the user with some feeling for the other likely classes.
  - Those situations are however not common and, in most applications, the maximum selection is made.
- That being so, the material presented about *decision surfaces* shows that the decision process has a geometric counterpart
  - in that a comparison of statistically derived discriminant functions leads equivalently to a *decision rule* that allows a pixel to be *classified on the basis of its position* in multispectral space compared with the location of a decision surface.
- This leads us to question whether a geometric interpretation can be adopted in general, without needing first to use statistical models.

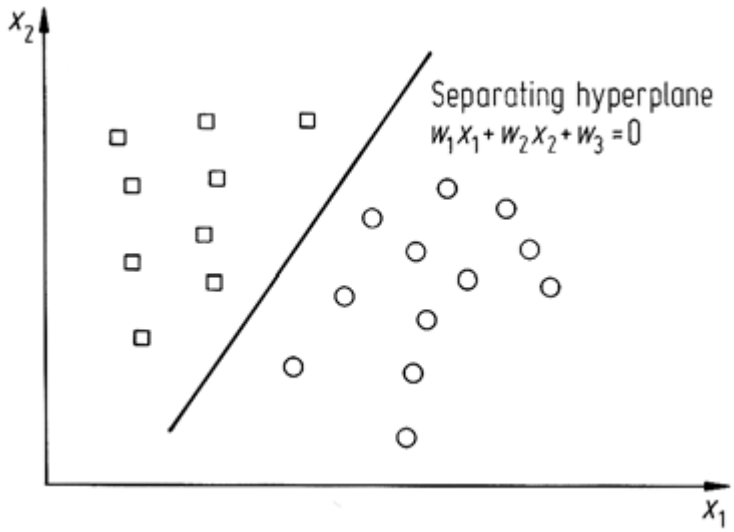
LINEAR DISCRIMINATION

Concept of a Weight Vector

- Consider the simple two class multispectral space shown in Figure, which has been constructed *intentionally* so that a simple straight line can be drawn between the pixels as shown.
- This straight line, which will be a multidimensional linear surface in general and which is called a ***hyperplane***, can function as a ***decision surface*** for classification.
- In the two dimensions shown, the equation of the line can be expressed

$$w_1 x_1 + w_2 x_2 + w_3 = 0$$

where the x_i are the *brightness value* co-ordinates of the multispectral space and the w_i are a set of coefficients, usually called ***weights***.



Concept of a Weight Vector

- There will be as many weights as the number of channels in the data, *plus one*.
- In general, if the number of channels or bands is *N*, the equation of a linear surface is

  $$w_1 x_1 + w_2 x_2 + \ldots + w_N x_N + w_{N+1} = 0$$

  which can be written as

  $$\boldsymbol{w}^t \boldsymbol{x} + w_{N+1} = 0 \qquad (8.22)$$

  where $\boldsymbol{x}$ is the *co-ordinate vector* and $\boldsymbol{w}$ is called the *weight vector*.
- In a real exercise the position of the separating surface would be unknown initially.
- Training a linear classifier amounts to determining an appropriate set of the weights that places the decision surface between the two sets of training samples.

- There is not necessarily a unique solution – any of an infinite number of (marginally different) decision hyperplanes will suffice to separate the two classes.
  - For a given data set, an explicit equation for the separating surface can be obtained using the minimum distance rule, as seen in minimum distance classification, which entails finding the mean vectors of the two class distributions.
  - An alternative method is outlined in the following, based on selecting an arbitrary surface and then iterating it into an acceptable position.
  - Even though not often used anymore, this method is useful to consider since it establishes some of the concepts used in *neural networks* and *support vector machines,* as we will see.

Testing Class Membership

- The calculation in (8.22) will be exactly zero only for values of $\boldsymbol{x}$ lying on the decision surface.
- If we substitute into that equation values of $\boldsymbol{x}$ corresponding to the pixel points indicated in the previous Figure, the left hand side will be non-zero.
- *For pixels in one class a positive result will be given, while pixels on the other side will give a negative result.*
- Thus, once the decision surface has been identified (i.e. trained), then a **decision rule is**

$$\begin{aligned} \boldsymbol{x} &\in \text{ class 1 if } \boldsymbol{w}^t\boldsymbol{x} + w_{N+1} > 0 \\ \boldsymbol{x} &\in \text{ class 2 if } \boldsymbol{w}^t\boldsymbol{x} + w_{N+1} < 0 \end{aligned} \tag{8.23}$$

Training

- A full discussion of linear classifier training is given in Nilsson (1965, 1990); only those aspects helpful to the neural network development following are treated here.

  It is expedient to define a new, augmented pixel vector according to

  $$\boldsymbol{y} = [\boldsymbol{x}^t, 1]^t$$

- If, in (8.22), we also take the term w_{N+1} into the definition of the weight vector, viz.

  $$\boldsymbol{w} = [\boldsymbol{w}^t, w_{N+1}]^t$$

  then the equation of the decision surface, can be expressed more compactly as

  $$\boldsymbol{w}^t \boldsymbol{y} = 0 \quad \text{(or equivalently } \boldsymbol{w} \cdot \boldsymbol{y} = 0\text{)}$$

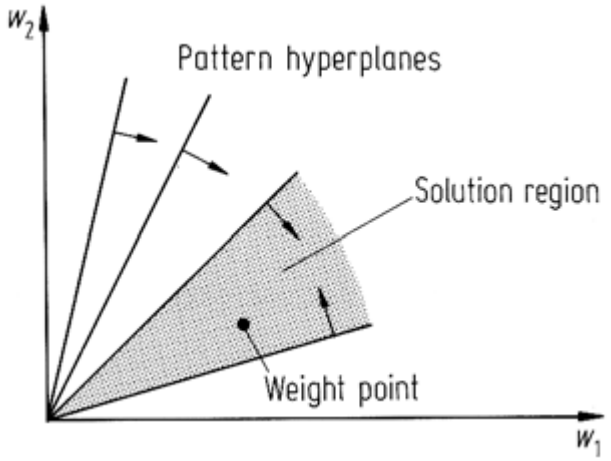
  so that the decision rule of (8.23) can be restated

  $$\begin{aligned} \boldsymbol{x} &\in \text{ class 1 if } \boldsymbol{w}^t \boldsymbol{y} > 0 \\ \boldsymbol{x} &\in \text{ class 2 if } \boldsymbol{w}^t \boldsymbol{y} < 0 \end{aligned} \tag{8.24}$$

- We usually think of $\boldsymbol{w}^t \boldsymbol{y} = 0$ as defining a linear surface in the $\boldsymbol{x}$ (or now $\boldsymbol{y}$) multispectral space, in which the coefficients of the variables (y_1, y_2, etc.) are the weights w_1, w_2, etc. However it is also possible to think of the equation as describing a linear surface in which the y's are the coefficients and the w's are the variables. This interpretation will see these surfaces plotted in a co-ordinate system which has axes w_1, w_2, etc.

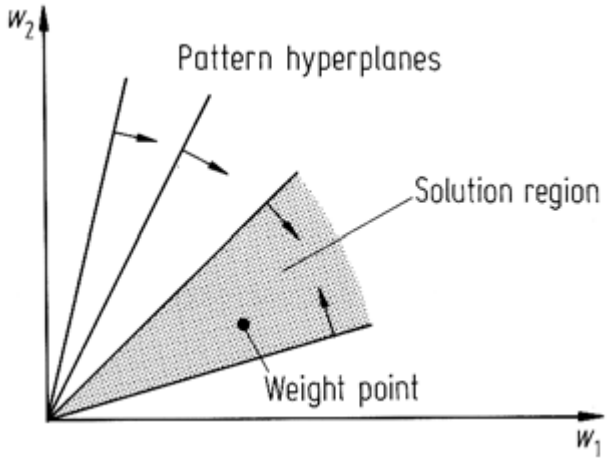
Training

- A two-dimensional version of this **weight space**, as it is called, is shown in Figure, in which have been plotted a number of pattern hyperplanes; these are specific linear surfaces in the new co-ordinates that pass through the origin and have, as their coefficients, the components of the (augmented) pixel vectors. Thus, while the pixels plot as points in multispectral space, they plot as linear surfaces in weight space.
- Likewise, a set of weight coefficients will define a surface in multispectral space, but will plot as a point in weight space. Although this is an abstract concept it will serve to facilitate an understanding of how a linear classifier can be trained.



Training

- In weight space the decision rule of (8.24) still applies – however *now it tests that the weight point is on the appropriate side of the pattern hyperplane*.
  - For example, Figure shows a single weight point which lies on the correct side of each pixel and thus defines a *suitable decision surface in multispectral space*. In the diagram, small arrows are attached to each pixel hyperplane to indicate the side on which the weight point must lie in order that the test of (8.24) succeeds for all pixels.
- The purpose of training the linear classifier is to ensure that the weight point is located somewhere within the solution region. If, through some **initial guess**, the weight point is located somewhere else in weight space then it ***has to be moved*** to the solution region.



Training

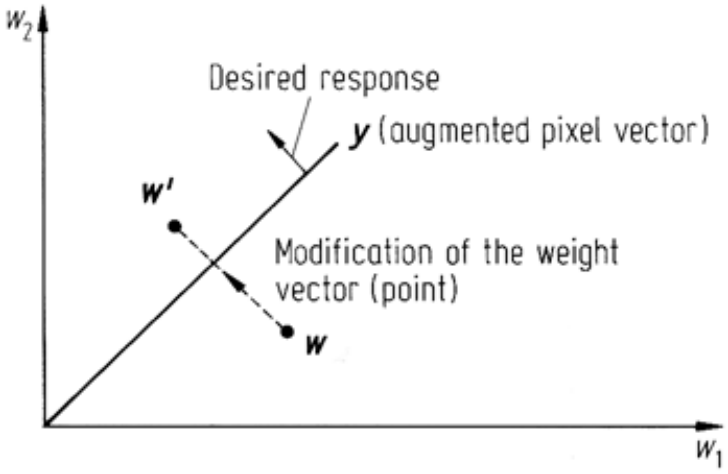
- Suppose an initial guess is made for the weight vector $\boldsymbol{w}$, but that this places the weight point on the wrong side of a particular pixel hyperplane as illustrated.
  - Clearly, the weight point *has to be shifted* to the other side to give a correct response in (8.24).
  - The most direct manner in which the weight point can be modified is to *move it straight across the pixel hyperplane*.
  - This can be achieved by adding a **scaled amount of the pixel vector** to the weight vector.

- The new position of the weight point is then

$$\boldsymbol{w}' = \boldsymbol{w} + c\boldsymbol{y} \qquad (8.25)$$

where c is called the **correction increment**, the size of which determines by how much the original weight point is moved orthogonal to the pixel hyperplane.

  - If it is large enough the weight point will be shifted right across the pixel plane, as required.



Training

- Having so modified the weight vector, the product in (8.24) then becomes

$$\begin{aligned} \boldsymbol{w}'^t \boldsymbol{y} &= \boldsymbol{w}^t \boldsymbol{y} + c \boldsymbol{y}^t \boldsymbol{y} \\ &= \boldsymbol{w}^t \boldsymbol{y} + c|\boldsymbol{y}|^2 \end{aligned}$$

  Clearly, if the initial $\boldsymbol{w}^t \boldsymbol{y}$ was erroneously negative a suitable positive value of c will give a positive value of $\boldsymbol{w}'^t \boldsymbol{y}$; otherwise a negative value of c will correct an erroneous initial positive value of the product.

- Using the class membership test in (8.24) and the correction formula of (8.25) the following iterative nonparametric training procedure, referred to as *error correction feedback*, is adopted.
  - First, an initial position for the weight point is chosen arbitrarily. Then, pixel vectors from training sets are presented one at a time. If the current weight point position classifies a pixel correctly then no action need be taken; otherwise the weight vector is modified as in (8.25) with respect to that particular pixel vector. This procedure is repeated for each pixel in the training set, and the set is scanned as many times as necessary to move the weight point into the solution region. If the classes are linearly separable then such a solution will be found.

Setting the Correction Increment

- Several approaches can be adopted for choosing the value of the correction increment, c. The simplest is to set c equal to a positive or negative constant (according to the change required in the $\boldsymbol{w}^t \boldsymbol{y}$ product). A common choice is to make $c = \pm 1$ so that application of (8.25) amounts simply to adding the augmented pixel vector to or subtracting it from the weight vector, thereby obviating multiplications and giving fast training.
- Another rule is to choose the correction increment proportional to the difference between the desired and actual response of the classifier:

$$c = \eta(t - \boldsymbol{w}^t \boldsymbol{y})$$

so that (8.25) can be written

$$\boldsymbol{w}' = \boldsymbol{w} + \Delta \boldsymbol{w}$$

with

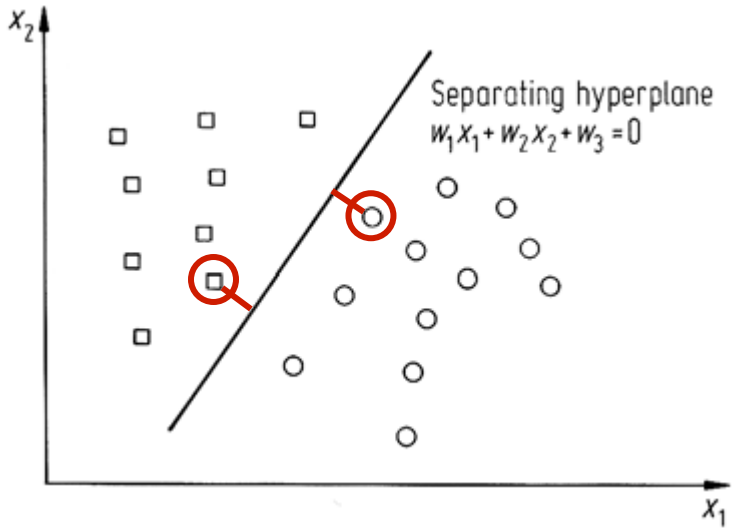
$$\Delta \boldsymbol{w} = \eta(t - \boldsymbol{w}^t \boldsymbol{y})\boldsymbol{y} \tag{8.26}$$

where t is the desired response to the training pattern $\boldsymbol{y}$ and $\boldsymbol{w}^t \boldsymbol{y}$ is the actual response; η is a factor which controls the degree of correction applied. Usually t would be chosen as $+1$ for one class and -1 for the other.

SUPPORT VECTOR CLASSIFIERS

Linearly Separable Data

- **The training process outlined for *linear discrimination* can lead to many, non-unique, yet acceptable solutions for the weight vector.**
  - The actual *size of the solution region in the weight space* (as seen) is an indication of that.
  - Also, *every training pixel takes part in the training process*; yet examination of the vector space suggests that it is **only** those *pixels in the vicinity of the separating hyperplane* that define where the hyperplane needs to lie in order to give a reliable classification.
- The **support vector machine (SVM)** provides a *training approach that depends only on those pixels in the vicinity of the separating hyperplane* (called the ***support pixel vectors***).
  - It also leads to a hyperplane position that is in a sense **optimal** for the available training patterns, as will be seen shortly.
  - SVM was invented by Vladimir Vapnik (1992)
  - The support vector concept was introduced to remote sensing image classification by Gualtieri and Cromp (1998). Two recent reviews that contain more detail than is given in the following treatment are by Burges (1998) and Huang et al. (2002). A very good recent treatment from a remote sensing perspective has been given by Melgani and Bruzzone (2004).



Linearly Separable Data

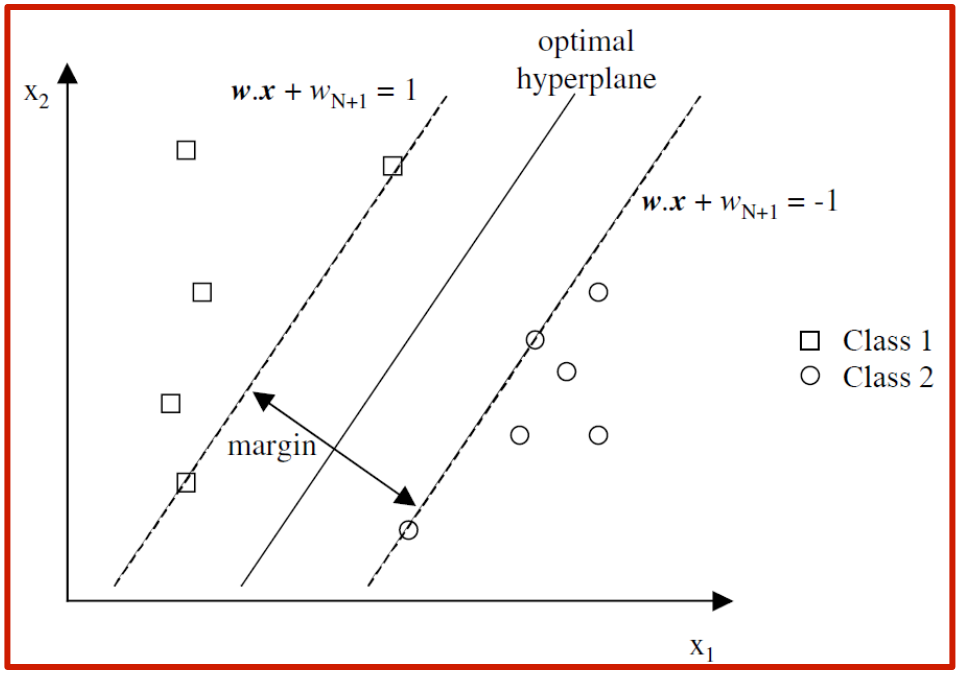
- If we expand the region in the vicinity of the hyperplane (see previous Figure) we can see, as suggested in Figure, that the optimal orientation of the hyperplane is when there is a **maximum separation** between the patterns in the two classes.
  - We can then draw two further hyperplanes **parallel** to the separating hyperplane, as shown, bordering the nearest training pixels from the two classes.
  - The equations for the hyperplanes are shown in the figure. Note that the *choice of unity* on the right hand side of the equations for the two marginal hyperplanes is *arbitrary* (related to a *common scale factor*), but it helps in the analysis.
  - If it were otherwise it could be scaled to unity by appropriately *scaling* the weighting coefficients w_k.

- Note that for pixels that lie *beyond the marginal hyperplanes*, we have
  - for class 1 pixels

$$\boldsymbol{w}.\boldsymbol{x} + w_{N+1} \geq 1 \qquad (8.28a)$$

  - for class 2 pixels

$$\boldsymbol{w}.\boldsymbol{x} + w_{N+1} \leq -1 \qquad (8.28b)$$



Linearly Separable Data

- It is useful now to describe the class label of the ith pixel by the variable y_i, which takes the value $+1$ for class 1 and -1 for class 2 pixels. Equations (8.28a) and (8.28b) can then be written as a single expression valid for pixels from both classes:

$$(\boldsymbol{w}.\boldsymbol{x} + w_{N+1})y_i \geq 1 \text{ for pixel } i \text{ in its correct class.}$$

- Alternatively

$$(\boldsymbol{w}.\boldsymbol{x} + w_{N+1})y_i - 1 \geq 0 \tag{8.29}$$

Equation (8.29) must hold for all pixels if the data is linearly separated by the two marginal hyperplanes of Figure. Those hyperplanes, defined by the equalities in (8.28), are described by

$$\boldsymbol{w}.\boldsymbol{x} + w_{N+1} - 1 = 0$$
$$\boldsymbol{w}.\boldsymbol{x} + w_{N+1} + 1 = 0$$

The perpendicular distances of these hyperplanes from the origin, respectively, are $-(w_{N+1} - 1)/||\boldsymbol{w}||$ and $-(w_{N+1} + 1)/||\boldsymbol{w}||$, where $\|\boldsymbol{w}\|$ is the Euclidean length of the weight vector. Therefore, the distance between the two hyperplanes, which is the *margin* in Figure, is $2/\|\boldsymbol{w}\|$.

Linearly Separable Data

- The best position (orientation) for the separating hyperplane will be that for which $2/\|\boldsymbol{w}\|$ is a maximum, or equivalently when the magnitude of the weight vector, $\|\boldsymbol{w}\|$, is a minimum. However there is a constraint! As we seek to maximise the margin between the two marginal hyperplanes by minimising $\|\boldsymbol{w}\|$ we must not allow (8.29) to be invalidated. In other words, all the training pixels must be on their correct side of the marginal hyperplanes. We handle the process of minimising $\|\boldsymbol{w}\|$ subject to that constraint by the process known as Lagrange multipliers. This requires us to set up a function (called the Lagrangrian) which includes the expression to be minimised ($\|\boldsymbol{w}\|$) from which is subtracted a proportion (α_i) of each constraint (one for each training pixel) in the following manner:

$$L = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i \{y_i(\boldsymbol{w}.\boldsymbol{x}_i + w_{N+1}) - 1\} \tag{8.30}$$

- The α_i are called Lagrange multipliers and are positive by definition, i.e.

$$\alpha_i \geq 0 \text{ for all } i.$$

Linearly Separable Data

- By minimising L we minimise $\|\boldsymbol{w}\|$ subject to the constraint (8.29).

  In (8.30) it is convenient to substitute

$$f(\boldsymbol{x}_i) = (\boldsymbol{w}.\boldsymbol{x}_i + w_{N+1})y_i - 1$$

  to give

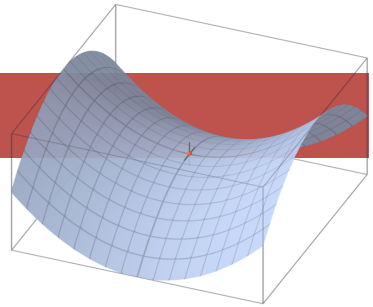
$$L = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i f(x_i)$$

  noting that for pixels in their correct class $f(\boldsymbol{x}_i) \geq 0$.

- It is useful here to remember what our task is. We have to find the *most* separated marginal hyperplanes in Fig. 8.16. In other words we need to find the $\boldsymbol{w}$ and w_{N+1} that minimises L and thus maximises the *margin* shown in the figure.

  But in seeking to minimise L (essentially during the training process) how do we treat the α_i? Suppose (8.29) is violated, as could happen for some pixels during training; then $f(\boldsymbol{x}_i)$ will be negative. Noting that α_i is positive that would cause L to increase. But we need to find values for $\boldsymbol{w}$ and w_{N+1} such that L is minimised.

Linearly Separable Data



- The worst possible case to handle is when the α_i are such as to cause L to be a maximum, since that forces us to minimise L with respect to $\boldsymbol{w}$ and w_{N+1} while the α_i are trying to make it as large as possible. The most robust approach to finding $\boldsymbol{w}$ and w_{N+1} (and thus the hyperplanes) therefore is to find the values of $\boldsymbol{w}$ and w_{N+1} that minimise L while simultaneously finding the α_i that try to maximise it.
- Thus we require, first, that:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$$

so that $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$. (8.31a)

- Secondly we require:

$$\frac{\partial L}{\partial w_{N+1}} = -\sum_i \alpha_i y_i = 0$$

so that $\sum_i \alpha_i y_i = 0$. (8.31b)

Linearly Separable Data

- Before proceeding, examine (8.30) again, this time for training pixels that satisfy the requirement of (8.29).
- What value(s) of α_i in (8.30) for those pixels maximise L?
- Since $y_i\ (\boldsymbol{w}.\boldsymbol{x}_i + w_{N+1}) - 1$ is now always positive then the only (non-negative) value of α_i that makes L as big as possible is $\alpha_i = 0$.
- Therefore, for any training pixels on the correct side of the marginal hyperplanes, $\alpha_i = 0$.
- This is an amazing, yet intuitive, result. It says we do not have to use any of the training pixel vectors, other than those that reside exactly on one of the marginal hyperplanes.
- The latter are called ***support vectors*** since they are the only ones that support the process of finding the marginal hyperplanes.
- Thus, in applying (8.31a) to find $\boldsymbol{w}$ we only have to use those pixels on the marginal hyperplanes.
- But the training is not yet finished! We still have to find the relevant α_i (i.e. those that maximise L and are non-zero).

Linearly Separable Data

- To proceed, note that we can put $\|\boldsymbol{w}\| = \boldsymbol{w}.\boldsymbol{w}$ in (8.30). Now (8.30), along with (8.31a), can be written most generally as:

$$
\begin{aligned}
L = \frac{1}{2}\left(\sum_i \alpha_i y_i \boldsymbol{x}_i\right) \cdot \left(\sum_j \alpha_j y_j \boldsymbol{x}_j\right) \\
- \sum_i \alpha_i \left[y_i \left(\left(\sum_j \alpha_j y_j \boldsymbol{x}_j \right) .\boldsymbol{x}_i + w_{N+1} \right) - 1 \right] \\
= \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i .\boldsymbol{x}_j - \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i .\boldsymbol{x}_j - w_{N+1} \sum_i \alpha_i y_i + \sum_i \alpha_i
\end{aligned}
$$

- Using (8.31b) this simplifies to

$$
L = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i .\boldsymbol{x}_j \tag{8.32}
$$

which has to be maximised by the choice of α_i. This usually requires a numerical procedure to solve for any real problem. Once we have found the α_i – call them α_i° – we can substitute them into (8.31a) to give the optimal training vector:

$$
\boldsymbol{w}^o = \sum_i \alpha_i^o y_i \boldsymbol{x}_i \tag{8.33a}
$$

Linearly Separable Data

- But we still do not have a value for w_{N+1}. Recall that on a marginal hyperplane

$$(\boldsymbol{w}.\boldsymbol{x}_i + w_{N+1})y_i - 1 = 0 \ .$$

Choose two support (training) vectors $\boldsymbol{x}(1)$ and $\boldsymbol{x}(-1)$ on each of the two marginal hyperplanes respectively for which $y = 1$ and -1. For these vectors we have

$$\boldsymbol{w}.\boldsymbol{x}(1) + w_{N+1} - 1 = 0$$

and

$$-\boldsymbol{w}.\boldsymbol{x}(-1) - w_{N+1} - 1 = 0$$

so that

$$w_{N+1} = \frac{1}{2}(\boldsymbol{w}.\boldsymbol{x}(1) + \boldsymbol{w}.\boldsymbol{x}(-1)) \ . \qquad (8.33b)$$

Normally sets of $\boldsymbol{x}(1)$, $\boldsymbol{x}(-1)$, would be used, with w_{N+1} found by averaging.

- With the values of α_i° determined by numerical optimisation, (8.33a,b) now give the parameters of the separating hyperplane that provides the largest margin between the two sets of training data. In terms of the training data, the equation of the hyperplane is:

$$\boldsymbol{w}^\circ.\boldsymbol{x} + w_{N+1} = 0$$

so that the discriminant function, for an unknown pixel $\boldsymbol{x}$ is

$$\boxed{g(\boldsymbol{x}) = \text{sgn}\ (\boldsymbol{w}^\circ.\boldsymbol{x} + w_{N+1})} \qquad (8.34)$$

Linear Inseparability – The Use of Kernel Functions

- If the pixel space is not linearly separable then the development of the previous section will not work without modification. A transformation of the pixel vector $\boldsymbol{x}$ to a different (usually higher order) feature space can be applied that renders the data linearly separable allowing the earlier material to be applied.
- The two significant equations for the linear support vector approach of the preceding section are (8.32) (for finding α_i°) and (8.34) (the resulting discriminant function). By using (8.33a), (8.34) can be rewritten

$$g(\boldsymbol{x}) = \text{sgn}\left\{\sum \alpha_i^\circ y_i \boldsymbol{x}_i.\boldsymbol{x} + w_{N+1}\right\} \tag{8.35}$$

- Now introduce the feature space transformation $\boldsymbol{x} \rightarrow \Phi(\boldsymbol{x})$ so that (8.32) and (8.35) become

$$L = \sum_i \alpha_i - 0.5 \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\boldsymbol{x}_i).\Phi(\boldsymbol{x}_j) \tag{8.36a}$$

$$g(\boldsymbol{x}) = \text{sgn}\left\{\sum \alpha_i^\circ y_i \Phi(\boldsymbol{x}_i).\Phi(\boldsymbol{x}) + w_{N+1}\right\} \tag{8.36b}$$

- In both (8.32) and (8.35) the pixel vectors occur only in the dot products. As a result the $\Phi(\boldsymbol{x})$ also appear only in dot products. So to use (8.36) it is strictly not necessary to know $\Phi(\boldsymbol{x})$ but only a scalar quantity equivalent to $\Phi(\boldsymbol{x}_i).\Phi(\boldsymbol{x}_j)$.

Linear Inseparability – The Use of Kernel Functions

- We call the product $\Phi(\boldsymbol{x}_i).\Phi(\boldsymbol{x}_j)$ a *kernel*, and represent it by $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ so that (8.36) becomes

$$L = \sum_i \alpha_i - 0.5 \sum_{i,j} \alpha_i \alpha_j y_i y_j k\left(\boldsymbol{x}_i.\boldsymbol{x}_j\right)$$

$$g(\boldsymbol{x}) = \text{sgn}\left\{\sum \alpha_i^\circ y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + w_{N+1}\right\}$$

Provided we know the form of the kernel we never actually need to know the underlying transformation $\Phi(\boldsymbol{x})$! Thus, after choosing $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ we then find the α_i° that maximise L and use that value in $g(\boldsymbol{x})$ to perform a classification.

- Two commonly used kernels in remote sensing are:

  The polynomial kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = [(\boldsymbol{x}_i).(\boldsymbol{x}_j) + 1]^p$

  The radial basis function kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$

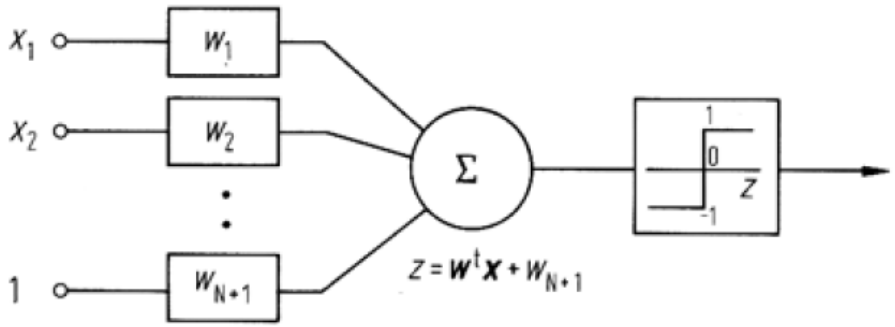
  in which p and γ are constants to be chosen.

- **Multicategory classification** The binary classifiers developed above can be used to perform multicategory classification by embedding them in a binary decision tree.

MULTICATEGORY CLASSIFICATION

Classification – The Threshold Logic Unit

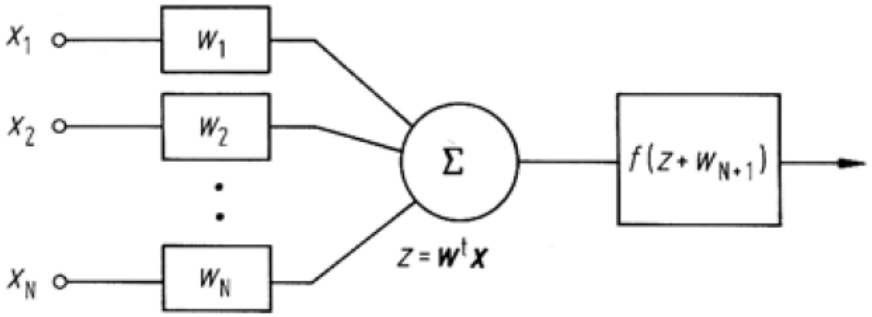
- After the linear, two category classifier has been trained, so that the final version of the weight vector $\boldsymbol{w}$ is available, it is ready to be presented with pixels it has not seen before in order to attach ground cover class labels to those pixels.
- This is achieved through application of the decision rule in (8.24).
- It is useful, in anticipation of neural networks, to *picture the classification rule* (8.24) *in diagrammatic form* as depicted in Figure **a**.
  - Simply, this consists of weighting elements, a summing device and an output element which, in this case, performs the maximum selection.
  - Together these are referred to as a **threshold logic unit** (**TLU**). It bears substantial similarity to the concept of a *processing element used in neural networks* for which the output thresholding unit is *replaced by a more general function* and the *pathway for the unity input* in the augmented pattern vector is actually *incorporated* into the output function.



a

Classification – The Threshold Logic Unit

- The latter can be done for a simple TLU as shown in Figure **b**, in which the simple thresholding element has been replaced by a functional block which performs the addition of the final weighting coefficient to the weighted sum of the input pixel components, and then performs a thresholding (or more general nonlinear) operation.
- Therefore a more useful representation of a processing element in which the thresholding function is generalised is depicted in the following Figure **b**

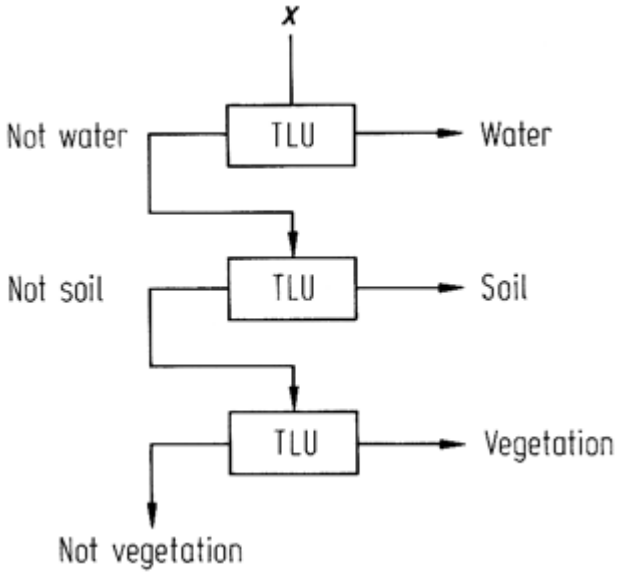


b

Multicategory Classification

- The foregoing work on linear classification has been based on an approach that can perform separation of pixel vectors into just two categories.
- Were it to be considered for remote sensing, it needs to be extended to be able to cope with a **multiclass problem**.
- Multicategory classification can be carried out in one of two ways.

1. First a **decision tree of linear classifiers** (TLUs) can be constructed, as seen in Figure, at each decision node of which a decision of the type (water or not water) is made.
   - At a subsequent node the (not water) category might be differentiated as (soil or not soil) etc. *It should be noted that the decision process at each node has to be trained separately*.



Multicategory Classification

2. Alternatively, a **multicategory version of the simple binary linear classifier** can be derived.
   - This reverts, for its derivation, to the concept of a **discriminant function** and, specifically, defines the **linear classifier discriminant function for class *i*** as

$$g_i(\boldsymbol{x}) = \boldsymbol{w}_i^t \boldsymbol{y} \qquad i = 1, \ldots M$$

   - Class membership is then decided on the basis of the *usual decision rule* expressed in (8.11a), i.e. according to the *largest of the* $g_i(\boldsymbol{x})$ for the given pixel vector $\boldsymbol{x}$.

- For **training**, an initial arbitrary **set of weight vectors** and thus discriminant functions is chosen. Then each of the training pixels is checked in turn.
  - Suppose, for a particular pixel the *j*th discriminant function is **erroneously largest**, **when in fact the pixel belongs** the *i*th category.
  - A **correction is carried out by adjusting the weight vectors for these two discriminant functions, to increase that for the correct class for the pixel and to decrease that for the incorrect class**, according to

$$\begin{aligned} \boldsymbol{w}_i' &= \boldsymbol{w}_i + c\,\boldsymbol{y} \\ \boldsymbol{w}_j' &= \boldsymbol{w}_j - c\,\boldsymbol{y} \end{aligned} \qquad (8.27)$$

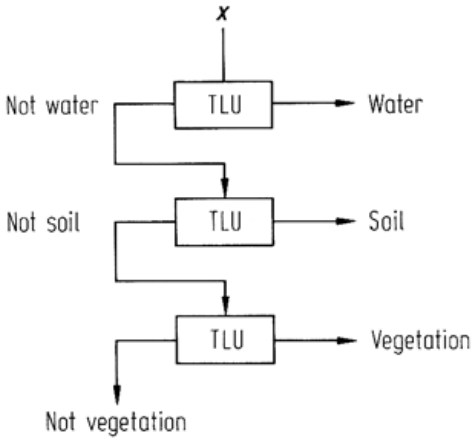
  where c is the correction increment.

- Again this correction procedure is iterated over the training set of pixels as many times as necessary to obtain a solution.
  - Nilsson (1965, 1990) shows that a solution is possible by this approach.

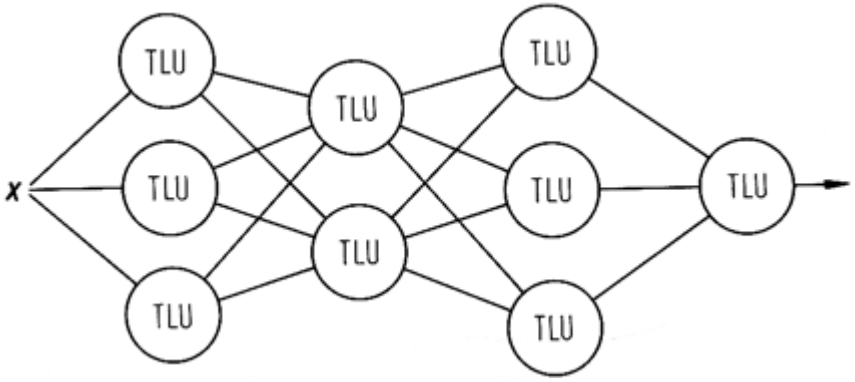
THE NEURAL NETWORK APPROACH

Networks of Classifiers – Solutions of Nonlinear Problems

- The already seen decision tree structure (Figure a) is a classifier network in that a collection of simple classifiers (in that case TLUs) is brought together to solve a complex problem.
- Nilsson (1965, 1990) has proposed a general **network structure** under the name of ***layered classifiers*** consisting entirely of interconnected TLUs, as shown in Figure b.
  - The benefit of forming a classifier network is that data sets that are inherently not separable with a simple linear decision surface should, in principle, be able to be handled since the layered classifier is known to be capable of implementing nonlinear surfaces.
  - The drawback however, is that training procedures for layered classifiers, consisting of *TLUs,* are difficult to determine.



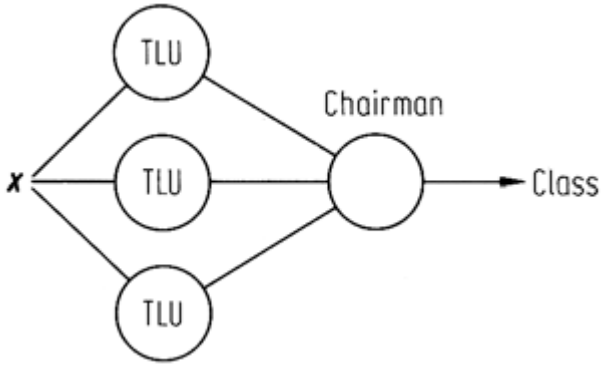
a



b

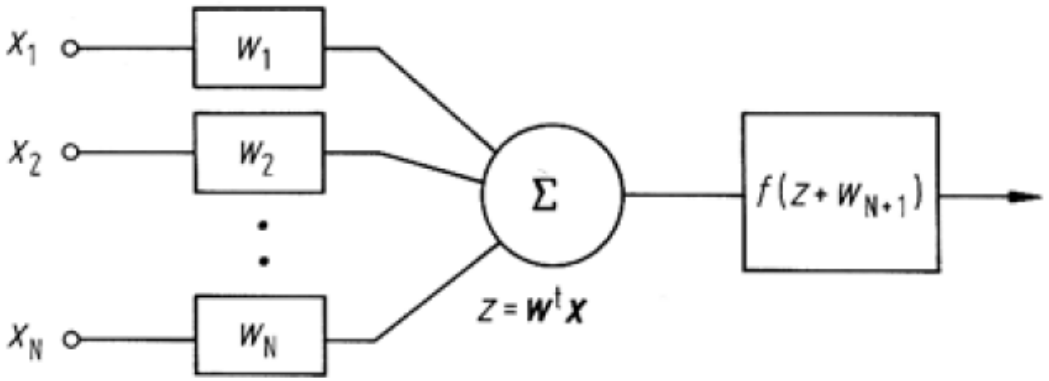
Networks of Classifiers – Solutions of Nonlinear Problems

- One specific manifestation of a layered classifier, known as a *committee machine*, is depicted in Figure.
  - Here the first layer consists simply of a set of TLUs, to each of which a pixel vector under test is submitted to see which of two classes is recommended.
  - The second layer is a single element which has the responsibility of judging the recommendations of each of the nodes in the first layer.
- It is therefore of the nature of a *chairman* or *vote taker*. It can make its decision on the basis of several sets of logic.
  - First, it can decide class membership on the basis of the majority vote of the first layer recommendations.
  - Secondly, it can decide on the basis of veto, in which all first layer classifiers have to agree before the vote taker will recommend a class.
  - Thirdly, it could use a form of seniority logic in which the chairman rank orders the decisions of the first layer nodes. It always refers to one first. If that node has a solution then the vote taker accepts it and goes no further. Otherwise it consults the next most senior of the first layer nodes, etc. (see Lee and Richards (1985)).



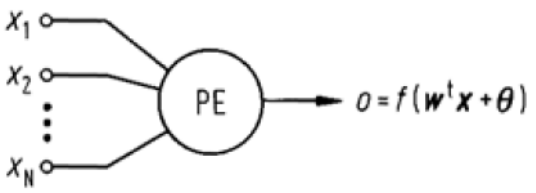
The Neural Network Approach

- For the purposes of this treatment a **neural network** is taken to be *of the nature of a layered classifier* such as depicted above, but with the *very important difference that* ***the nodes are not TLUs***, *although resembling them closely*.
  - **The node structure in Figure can be made much more powerful, and coincidentally lead to a training theorem for multicategory nonlinear classification, if the output processing element does not apply a thresholding operation to the weighted input but rather applies a softer, and mathematically differentiable, operation.**



The Processing Element

- The essential processing node in the neural network to be considered here (sometimes called a **neuron** by analogy to biological data processing from which the term neural network derives) is an element, as anticipated previously, with *many inputs and with a single output*, depicted simply in Figure.



- Its operation is described by

$$o = f(\mathbf{w}^t\mathbf{x} + \theta) \tag{8.37}$$

where θ is a threshold (sometimes set to zero), $\mathbf{w}$ is a vector of weighting coefficients and $\mathbf{x}$ is the vector of inputs.

  - For the special case when the inputs are the band values of a particular multispectral pixel vector it could be envisaged that the threshold θ takes the place of the weighting coefficient w_{N+1} in (8.22).
  - If the function f is a thresholding operation this processing element would behave as a TLU.
  - In general, the number of inputs to a node will be defined by network topology as well as data dimensionality, as will become evident.

- The major difference between the layered classifier of TLUs shown before and the neural network, known as the **multilayer perceptron**, is in the choice of the function f, called the **activation function**.
  - Its specification is simply that it emulate thresholding in a soft or asymptotic sense and be differentiable.

The Processing Element

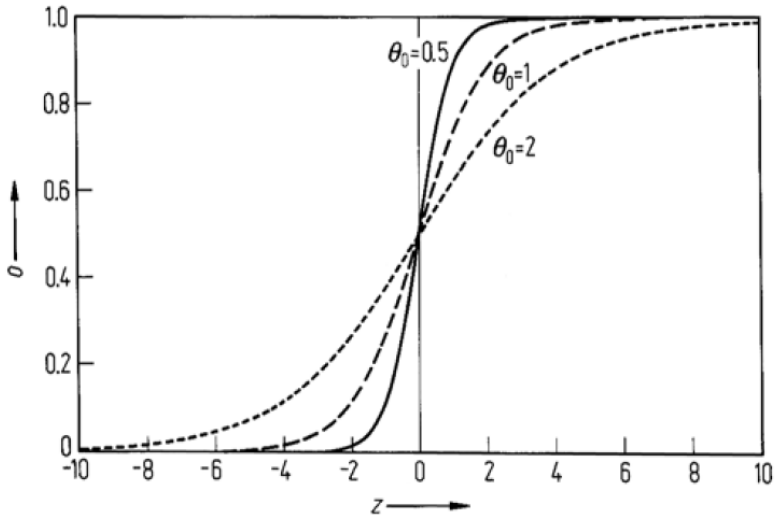
- The most commonly encountered expression is

$$f(z) = \frac{1}{1 + e^{-z/\theta_0}} \tag{8.38}$$

where the argument z is $\boldsymbol{w}^t\boldsymbol{x} + \theta$ as seen in (8.37) and θ_0 is a constant.

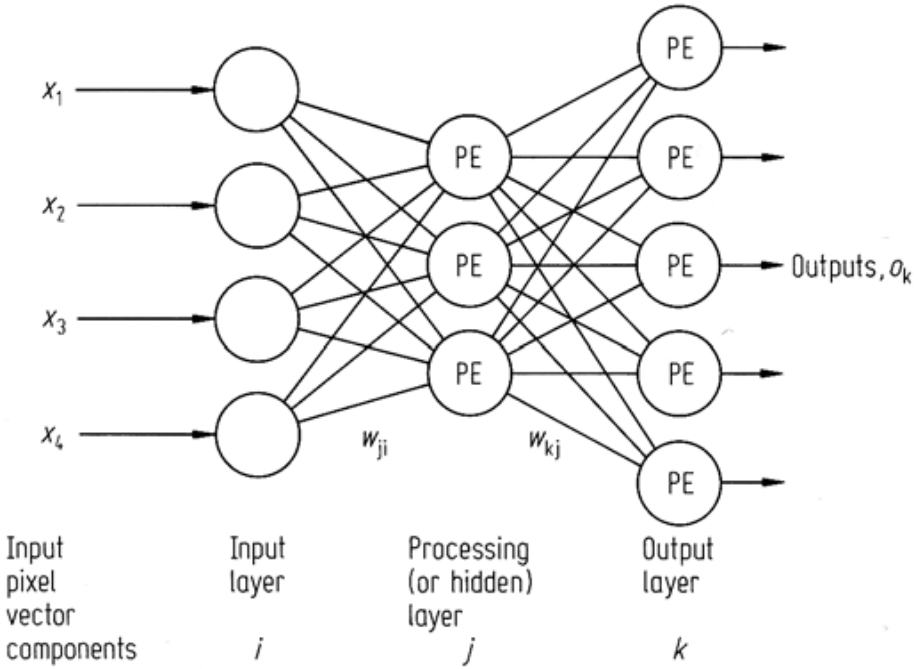
  - This approaches 1 for z large and positive and 0 for z large and negative and is thus *asymptotically thresholding*.
  - It is important to recognise that the outcome of the product $\boldsymbol{w}^t\boldsymbol{x}$ is a simple scalar

- When plotted with $\theta = 0$, (8.38) appears as shown in Figure.
  - For θ_0 very small the activation function approaches a thresholding operation.
  - Usually $\theta_0 = 1$.



The Processing Element

- A **neural network for use in remote sensing image analysis** will appear as shown in Figure, being a *layered classifier composed of processing elements of the type shown before*.
  - It is conventionally drawn with an *input layer of nodes* (which has the function of *distributing* the inputs to the processing elements of the next layer, and *scaling* them if necessary) and an *output layer* from which the class labelling information is provided.
  - In between there may be *one or more so-called hidden or other processing layers* of nodes. Usually one hidden layer will be sufficient, although the number of nodes to use in the hidden layer is often not readily determined. We return to this below.



Training the Neural Network – Backpropagation

- Before it can perform a classification, the network must be trained.
  - This amounts to using labelled training data to help determine the weight vector $\boldsymbol{w}$ and the threshold θ in (8.37) for each processing element connected into the network.
  - Note that the *constant* θ_0 in (8.38), which governs the gradient of the activation function as seen in a previous Figure, is generally pre-specified and *does not need to be estimated* from the training data.
- Part of the complexity in understanding the training process for a neural net is caused by the *need to keep careful track of the parameters and variables over all layers and processing elements*, how they vary with the presentation of training pixels and (as it turns out) with iteration count.
  - This can be achieved with a detailed *subscript convention*, or by the use of a simpler *generalised notation*.
  - We will adopt the latter approach, following essentially the development by Pao (1989).
- The derivation will be focussed on a 3 layer neural net, since this architecture has been found sufficient for many applications.
  - However the results *generalise to more layers*.

Training the Neural Network – Backpropagation

- The previous Figure incorporates the *nomenclature* used.
  - The three layers are lettered as i, j, k with k being the output.
  - The set of *weights* linking layer i PEs with those in layer j are represented generally by w_{ji}, while those linking layers j and k are represented by w_{kj}.
  - There will be a very large number of these weights, but in deriving the training algorithm it is *not necessary to refer to them all individually*.
  - Similarly the general activation function *arguments* z_i and *outputs* o_i, can be used to represent all the arguments and outputs in the corresponding layer.
- For j and k layer PEs (8.37) is

$$o_j = f(z_j) \quad \text{with} \quad z_j = \sum_i w_{ji} o_i + \theta_j \tag{8.39a}$$

$$o_k = f(z_k) \quad \text{with} \quad z_k = \sum_j w_{kj} o_j + \theta_k \tag{8.39b}$$

- The sums in (8.39) are shown with respect to the indices i and j.
  - This should be read as meaning the sums are taken over all inputs of particular layer j and layer k PEs respectively.
  - Note also that the sums are expressed in terms of the outputs of the previous layer since these outputs form the inputs to the PEs in question.

Training the Neural Network – Backpropagation

- An *untrained* or poorly trained network will give *erroneous* outputs.
  - Therefore, as a *measure of how well a network is functioning* during training, we can assess the outputs at the last layer (k).
  - A suitable measure along these lines is to use the sum of the *squared output error*.
  - The error made by the network when presented with *a single training pixel* can thus be expressed

$$E = \frac{1}{2}\sum_{k}(t_k - o_k)^2 \qquad (8.40)$$

  where the t_k represent the desired or target outputs(*) and o_k represents the actual outputs from the output layer PEs in response to the training pixel.
  - The factor of ½ is included for arithmetic convenience in the following.
  - The sum is over all output layer PEs.
- A useful training strategy is to adjust the weights in the processing elements until the error has been minimised, at which stage the actual outputs are as close as possible to the desired outputs.

(*) These will be specified from the training data labelling. The actual value taken by t_k however will depend on how the outputs themselves are used to represent classes. Each output could be a specific class indicator (e.g. 1 for class 1 and 0 class 2); alternatively some more complex coding of the outputs could be adopted. This is considered following.

Training the Neural Network – Backpropagation

- A common approach for adjusting weights to reduce (and thus minimise) the value of a function of which they are arguments, is to modify their values proportional to the negative of the partial derivative of the function. This is called a **gradient descent technique** (**). Thus for the weights linking the j and k layers let

$$w'_{kj} = w_{kj} + \Delta w_{kj}$$

with

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}}$$

where η is a positive constant that controls the amount of adjustment. This requires an expression for the partial derivative, which can be determined using the chain rule

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial z_k} \frac{\partial z_k}{\partial w_{kj}} \qquad (8.41)$$

each term of which must now be evaluated.

(**) Another optimisation procedure used successfully for neural network training in remote sensing is the *conjugate gradient method* (Benediktsson et al., 1993).

Training the Neural Network – Backpropagation

- From (8.39b) and (8.38) we see (for $\theta_0 = 1$)

$$\frac{\partial o_k}{\partial z_k} = f'(z_k) = (1 - o_k)o_k \tag{8.42a}$$

and

$$\frac{\partial z_k}{\partial w_{kj}} = o_j \tag{8.42b}$$

Now from (8.40)

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{8.42c}$$

Thus the correction to be applied to the weights is

$$\Delta w_{kj} = \eta(t_k - o_k)(1 - o_k)o_k o_j \tag{8.43}$$

For a given trial, all of the terms in this expression are known so that a beneficial adjustments can be made to the weights which link the hidden layer to the output layer.

Training the Neural Network – Backpropagation

- Now consider the weights that link the i and j layers. The weight adjustments are

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}}$$

In a similar manner to the above development we have

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial o_j} (1 - o_j) o_j o_i$$

Unlike the case with the output layer, however, we cannot obtain an expression for the remaining partial derivative from the error formula, since the o_j are not the outputs at the final layer, but rather those from the hidden layer. Instead we express the derivative in terms of a chain rule involving the output PEs. Specifically

$$\begin{aligned} \frac{\partial E}{\partial o_j} &= \sum_k \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial o_j} \\ &= \sum_k \frac{\partial E}{\partial z_k} w_{kj} \end{aligned}$$

The remaining partial derivative can be obtained from (8.42a) and (8.42c) as

$$\frac{\partial E}{\partial z_k} = -(t_k - o_k)(1 - o_k) o_k$$

so that

$$\Delta w_{ji} = \eta (1 - o_j) o_j o_i \sum_k (t_k - o_k)(1 - o_k) o_k w_{kj} \tag{8.44}$$

Training the Neural Network – Backpropagation

- Having determined the w_{kj} from (8.43), it is now possible to find values for the w_{ji} since all other entries in (8.44) are known or can be calculated readily.

For convenience we now define

$$\delta_k = (t_k - o_k)(1 - o_k)o_k \tag{8.45a}$$

and

$$\delta_j = (1 - o_j)o_j \sum_k (t_k - o_k)(1 - o_k)o_k w_{kj}$$

$$= (1 - o_j)o_j \sum_k \delta_k w_{kj} \tag{8.45b}$$

so that we have

$$\Delta w_{kj} = \eta \delta_k o_j \tag{8.46a}$$

and

$$\Delta w_{ji} = \eta \delta_j o_i \tag{8.46b}$$

both of which should be compared with (8.26) to see the effect of a differentiable activation function.

The thresholds θ_j and θ_k in (8.39) are found in exactly the same manner as for the weights in that (8.46) is used, but with the corresponding inputs chosen to be unity.

Training the Neural Network – Backpropagation

- Now that we have the mathematics in place it is possible to describe how training is carried out.
  - The network is initialised with an arbitrary set of weights in order that it can function to provide an output.
  - The training pixels are then presented one at a time to the network.
  - For a given pixel the output of the network is computed using the network equations.
- Almost certainly the output will be incorrect to start with – i.e. the o_k will not match the desired class t_k for the pixel, as specified by its labelling in the training data.
  - Correction to the output PE weights, described in (8.46a), is then carried out, using the definition of δ_k in (8.45a).
  - With these new values of δ_k and thus w_{kj} (8.45b) and (8.46b) can be applied to find the new weight values in the earlier layers.
  - In this way the effect of the output being in error is *propagated back through the network* in order to correct the weights.
- The technique is thus often referred to as **back propagation**.

Training the Neural Network – Backpropagation

- Pao (1989) recommends that the weights not be corrected on each presentation of a single training pixel, but rather that the *corrections for all pixels in the training set be aggregated into a single adjustment*.
- Thus for *p* training patterns the bulk adjustments are

$$\Delta w'_{kj} = \sum_p \Delta w_{kj} \qquad \text{and} \qquad \Delta w'_{ji} = \sum_k \Delta w_{ji}$$

This is tantamount to deriving the algorithm with the error being calculated over all pixels p in the training set, viz $E_p = \sum_p E$, where E is the error expressed for a single pixel in (8.40).

- After the weights have been so adjusted the *training pixels are presented to the network again* and the outputs re-calculated to see if they correspond better to the desired classes.
- *Usually they will still be in error and the process of weight adjustment is repeated*.
  - Indeed the process is iterated as many times as necessary in order that the network respond with the correct class for each of the training pixels or until the number of errors in classifying the training pixels is reduced to an acceptable level.

Choosing the Network Parameters

- When considering the use of the neural network approach to classification it is necessary to make several key decisions beforehand.
  - First, the **number of layers** to use must be chosen.
    - gGenerally, a *three layer network is sufficient*, with the purpose of the first layer being simply to distribute (or fan out) the components of the input pixel vector to each of the processing elements in the second layer.
    - Thus the first layer does no processing as such, apart perhaps from *scaling* the input data, if required.
  - The next choice relates to the **number of elements in each layer**.
    - The *input layer* will generally be given as many nodes as there are components (features) in the pixel vectors.
    - The number to use in the *output node* will depend on how the outputs are used to represent the classes.
      - The simplest method is to let each *separate output signify a different class*, in which case the number of output processing elements will be the same as the number of training classes.
      - Alternatively, a *single PE could be used to represent all classes*, in which case a different value or level of the output variable will be attributed to each class.
      - A further possibility is to *use the outputs as a binary code*, so that two output PEs can represent four classes, three can represent 8 classes and so on.
    - As a general guide the number of PEs to choose for the *hidden or processing layers* should be *the same as or larger than the number of nodes in the input layer* (Lippmann, 1987).