Acta Numerica (1994), *pp.* 145–202

Aspects of the numerical analysis of neural networks

S.W. Ellacott
Department of Mathematical Sciences
University of Brighton
Moulsecoomb, Brighton BN2 4GJ
England
E-mail: swe@unix.bton.ac.uk

This article starts with a brief introduction to neural networks for those unfamiliar with the basic concepts, together with a very brief overview of mathematical approaches to the subject. This is followed by a more detailed look at three areas of research which are of particular interest to numerical analysts.

The first area is approximation theory. If K is a compact set in $\mathbb{R}^n$, for some n, then it is proved that a semilinear feedforward network with one hidden layer can uniformly approximate any continuous function in $C(K)$ to any required accuracy. A discussion of known results and open questions on the degree of approximation is included. We also consider the relevance of radial basis functions to neural networks.

The second area considered is that of learning algorithms. A detailed analysis of one popular algorithm (the delta rule) will be given, indicating why one implementation leads to a stable numerical process, whereas an initially attractive variant (essentially a form of steepest descent) does not. Similar considerations apply to the backpropagation algorithm. The effect of filtering and other preprocessing of the input data will also be discussed systematically.

Finally some applications of neural networks to numerical computation are considered.

CONTENTS

1. An introduction to neural networks

1.1. A network to compute 'XOR'

A neural network is a model of computation based loosely on the mammalian brain. Rather than give a formal definition we illustrate by a simple example. Figure 1 shows a network designed to compute the 'exclusive-or' (XOR) function. Each input unit takes a single scalar input. In general these may take any real value, but for this particular example the inputs are restricted to the values 0 or 1. Thus the set of possible input vectors is

$$\{(0,0)^T, (0,1)^T, (1,0)^T, (1,1)^T\}.$$

The network is required to compute the output '0' if the two inputs are the same or '1' if they are different. It does this in the following way. The vertices of the graph shown as circles are called *units* or *neurons*. The input units shown with no inscribed numbers simply pass their inputs to their output edges which in this context are called *links* or *synapses*. They are multiplied by the *weights* shown on these links and summed at the input to the next unit. Suppose for instance the input vector is $(1,0)^T$. The input to the unit inscribed '1.5' is thus $1 \times 1 + 1 \times 0 = 1$. In this type of network, the '1.5' itself is a threshold value. Since the total input $1 < 1.5$, the output of the unit is 0. If the original input vector were $(1,1)^T$, the input to this unit would be $2 > 1.5$, so this unit would output 1. The output of a unit as a function of the given input is called the *activation function* of the unit. With the original input vector $(1,0)^T$, we see that the input to the unit

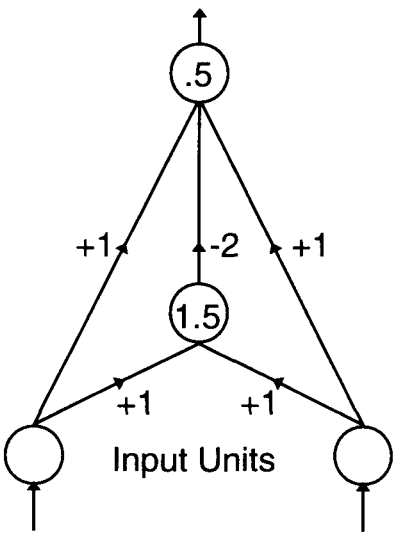

Fig. 1.

inscribed '.5' is thus $1 \times 1 - 2 \times 0 + 1 \times 0 = 1$. Since $1 > .5$, the output of the whole network is 1 as required. The reader might like to verify in a similar manner that the network correctly computes the exclusive-or of the other three possible input vectors.

In this case the network is being used as a pattern classifier: the input patterns are separated into classes. For this example there are just two classes according to whether the XOR of the inputs is 0 or 1. Thus a single binary output is sufficient. In other cases we may require more than two classes. This can be achieved by allowing the output to take more values or, alternatively by using more than one output. Originally neural nets were thought of as logical devices like conventional computers. Pattern classification is thus the classical application. However, it is now becoming clear that much of that phenomenon that we think of as intelligence is not really binary in this way. A tennis player attempting to hit a ball is required to produce a complex muscular response to equally complex visual, tactile and kinesthetic input data. The problem is one of control theory at a complicated mutlivariate level. The human nervous system and brain in this context is best thought of as a nonlinear analogue controller with learning capability. Control applications of neural nets constitute a major application area (see e.g. Warwick *et al.*, 1992; Werbos, 1992), as do pattern recognition problems in speech and vision where it is difficult to get good models to analyse conventionally (e.g. Linggard and Nightingale, 1992). The ability of neural systems to 'learn' (see Section 3) means that they can produce usable models on the basis of examples *without* the need of a formal axiomatic analysis.

Humans can also recognize patterns and structure in data when the required output is not known. In statistical language, the problem then is one of clustering rather than classification. This is rather more difficult than simply learning known patterns, but it can also be tackled by neural networks. The most popular network for this problem is that due to Kohonen (see, for example, Wasserman (1990) Ch. 4). We will not discuss the Kohonen net here, but note that the mathematical problems are somewhat similar to the networks we do consider.

There is a vast range of variations on these general ideas: the interested reader should consult one of the many textbooks available on this subject. For an introductory treatment, see Aleksander and Morton (1990), Simpson (1990) or Wasserman (1989). A deeper work, although now a little dated, is Rumelhart and McClelland (1986).

1.2. Perceptrons and multilayer perceptrons

In this article we shall concentrate on the simplest of all neural networks, the *perceptron*, and an extension of this called the *multilayer perceptron*

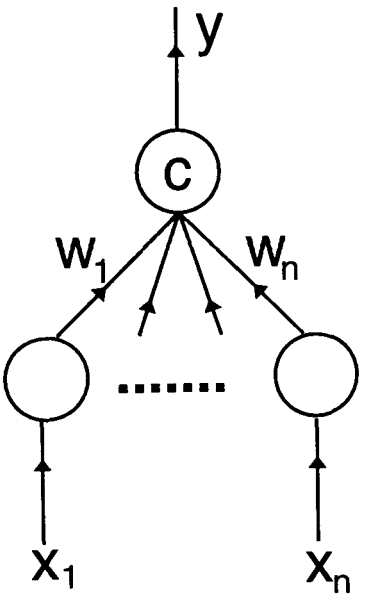

Fig. 2.

or *semilinear feedforward network.* The density results of Section 2 refer to the latter. It is the most popular of all neural network architectures, probably because it is relatively easy to understand and use. (However, many of the ideas are much more widely applicable and the formulation of the backpropagation algorithm given in Section 3 is certainly much more general than is required just for the multilayer perceptron.) We will look briefly at some other architectures including the Hopfield net in Section 4.

Figure 2 shows a simple perceptron with a single output. The units are interpreted as in Figure 1 with the input units having identity activation function and the output unit having a simple threshold. Thus denoting the input vector by $\mathbf{x}$ and the weight vector by $\mathbf{w}$ (both in $\mathbb{R}^n$) it is easy to see that the output y is 1 if $\mathbf{w}^T\mathbf{x} > c$ and 0 if $\mathbf{w}^T\mathbf{x} \leq c$. So for a fixed weight vector $\mathbf{w}$ and threshold c, the network divides $\mathbb{R}^n$ into two half spaces separated by a hyperplane. For obvious reasons, the perceptron is described as a linear network. Considering the case $n = 2$, observe that the pairs of inputs required to produce outputs 1 or 0 for the exclusive-or function are at diagonally opposite corners of a square *so they cannot be separated by a perceptron.* This simple observation delayed the development of neural networks for many years until tools for handling nonlinear networks such as Figure 1 became available.

In spite of this restriction, it is worth studying the perceptron as it constitutes the simplest case of many other network architectures and can give very useful insight into their behaviour. The multilayer perceptron shown in

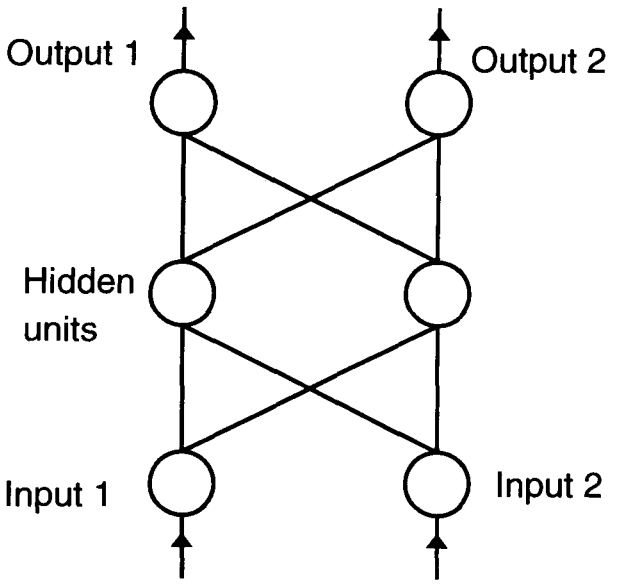

Fig. 3.

Figure 3 is the most obvious and straightforward generalization. It consists simply of layers of perceptrons connected together in cascade. Observe that the AND and OR logical functions *can* be separated linearly. If we wish to classify sets of points into two sets, we can certainly do it with three perceptrons in cascade: the first layer divides the plane into half-planes, the second can AND these to produce polygons and the third can OR these polygons to assign them to a required class. We can therefore construct any desired partition of $\mathbb{R}^n$ into polygonal regions, and classify these regions. This process is illustrated for the two-dimensional case in Figure 4. Originally it was believed that this was the minimum number of layers needed to solve the classification problem, but it is now known that, in principle, two perceptrons in cascade (i.e. only one hidden layer) is sufficient. A proof of this will be given in Section 2. However, it does seem plausible to suppose that nets with two or more hidden layers might be more efficient. This question is still open.

In Figure 3 we have shown just two neurons in each layer. In practice the input layer must match the dimension of the input vectors and the output layer provides the number of desired outputs (usually small). However, the number of units in the hidden layers may be chosen by the designer. For the present we may still regard the units as having thresholds although in fact this is not usually the way they are implemented, as will be described in Section 2.

1.3. Mathematical approaches to neural networks

The theory of conventional computing devices is mostly a matter of discrete mathematics. Indeed, computation has inspired considerable advances in this branch of mathematics (Taylor, 1993). However as we have already seen, neural nets can be considered as analogue devices so the required mathematics is much more classical. The problems of classification and clustering traditionally belong to the statisticians. The structure of the classification space can be analysed using statistical decision theory (Amari, 1990). If we attempt to understand the actual behaviour of biological neurons we are in the realm of mathematical modelling. For example, some authors have considered architectures involving coupled oscillators and/or chaos theory (Jones, 1992). Neurons can be thought of as simple, statistically identical 'particles' linked by the synapses. Thus the large-scale behaviour of assemblies of neurons has much in common with the statistical physics of gasses (Venkataraman and Athithan, 1991). Dynamic behaviour can be introduced into networks in many different ways. Some network architectures are recursive (consider the output of a multilayer perceptron being fed back in as part of the input, or the Hopfield net introduced in Section 4 below). Others are defined by or approximated by differential equations. In fact the whole panoply of dynamical systems and control theory underlies the study or neural networks in a manner too pervasive to be adequately surveyed here. Differential topology has its adherents (Wang *et al.*, 1992). Indeed the theory of neural networks would appear to be almost as chaotic as their dynamic behaviour. It seems certain many of the results must be duplicated in different papers under different names and using different languages. There is a real need for the subject to develop its own coherent structure, rather than borrowing from a host of other mathematical disciplines. Notwithstanding this remark, we will now proceed to investigate neural networks from the viewpoint of numerical analysis!

2. Density and approximation by neural networks

A natural question arising from the ideas developed in Section 1 is to consider what sets of points a given network can classify. We have already seen that a multilayer perceptron can separate any finite sets of points in $\mathbb{R}^n$. Let us describe this a little more carefully. Let A and B be two finite sets in $\mathbb{R}^n$. In Figure 4, A might consist of the points labelled o and B the points labelled χ. Suppose we wish the network to produce output 1 for points in A and 0 for points in B. Clearly it is possible to construct a finite set of polygons $P_1 \cdots P_k$ such that

$$A \subset Q := \cup P_j \quad \text{and} \quad B \cap P_j = \phi \quad \text{for } j = 1, \ldots, k. \tag{2.1}$$

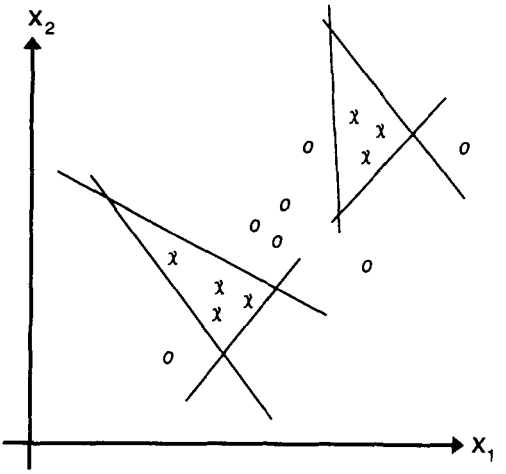

Fig. 4.

Each P_j consists of a finite intersection of half spaces. It can thus be obtained by a network computing the logical AND function which is linearly separable. The union to include A can then be obtained by a network computing the OR function: OR is also linearly separable. This approach is natural and simple, but it is difficult to take it very far. Moreover it only applies to discrete logical functions. We would like our networks to be able to cope with continuous problems such as the control problems involved in balancing a rocket or catching a ball. A different viewpoint proves more fruitful.

As before, we regard our inputs as vectors in $\mathbb{R}^n$. The output $\mathbf{y}$ of the network is a vector in $\mathbb{R}^m$ where usually $m \ll n$. (In many cases $m = 1$.) The network thus computes a function $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$ which we regard as an approximation to some other function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$. The point classification problem discussed above can be put into this context by choosing $m = 1$ and $\mathbf{f}$ to be the characteristic function of the set Q in (2.1) (i.e. $\mathbf{f}(x) = 1$ if $x \in Q$ and 0 otherwise). This viewpoint means that neural networks can be discussed using methods derived from approximation theory. The point sets A and B are conveniently regarded as interpolation or sample points for approximation of the function $\mathbf{f}$. (Sometimes networks are actually constructed this way: the radial basis function networks discussed in Section 2.3 are of this type (Broomhead and Lowe, 1988; Mason and Parks, 1992).) As constructed here the function $\mathbf{f}$ is not continuous; however since the point sets A and B are finite, it is clearly possible to overcome this with some smoothing process.

The question of what a neural net can compute may thus be restated in approximation theoretic terms. Specifically we wish to know if our set

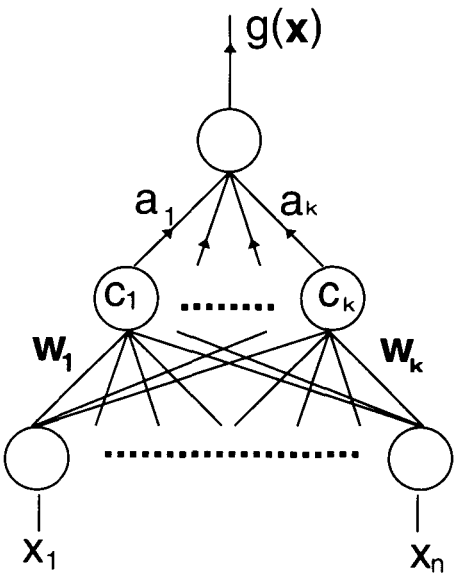

Fig. 5.

of possible functions g, corresponding to our particular class of network, is dense in some suitable function space which includes our target function f. This question is the subject of this section. In fact nearly all the results just consider the case of a single output network $m = 1$, so we will also make this simplification. Our functions are thus (scalar) real valued, so we drop the vector bold type and just refer to them as g and f.

In view of the difficulty of dealing with nondifferentiable and discontinuous functions, it is usual to use a smooth activation function, instead of a threshold, for the units. (The activation function was explained in section 1.1. It is the function that relates the sum of the inputs to a given unit to the output.) Note that the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ Various restrictions need to be put on σ to make a practical network, but we will introduce these as required.

The results to be discussed next refer to the case of an multilayer perceptron with a *single* hidden layer of k units. The form of the network is shown in Figure 5. The weight vector relating the inputs to the jth hidden neuron is denoted $\mathbf{w}_j$. Thus for a given input $\mathbf{x}$, the input to this unit is $\mathbf{w}_j^T\mathbf{x}$. We assume that each of the hidden units has identical activation function σ, but that a 'threshold like' shift of the argument by a real scalar c_j is permitted. So the output from the jth hidden unit is

$$\sigma(\mathbf{w}_j^T\mathbf{x} + c_j).$$

Functions of this form are called *ridge functions*. The name derives from the fact that they are obviously constant on the hyperplane $\mathbf{w}_j^T\mathbf{x} =$ constant. In two dimensions, this means that the contours of the function form straight

ridges. An immediate consequence of this observation is the fact that no nontrivial continuous ridge function can be in $L_1(\mathbb{R}^n)$ if $n > 1$. To see this, choose $\mathbf{x}_0$ for which $\sigma(\mathbf{w}_j^T\mathbf{x}_0 + c_j) \neq 0$. Then integrate $|\sigma|$ over the infinite strip $|\mathbf{w}_j^T\mathbf{x} - \mathbf{w}_j^T\mathbf{x}_0| < \delta$, where δ is chosen sufficiently small that σ does not vanish in the region.

Now we denote the weight connecting the jth hidden unit to the output by a_j. The output function g of the network is therefore

$$g(\mathbf{x}) = \sum_j^k a_j \sigma(\mathbf{w}_j^T\mathbf{x} + c_j). \tag{2.2}$$

Activation functions σ used in practice have the property of being monotonic increasing, bounded and *sigmoidal*, which means that the limits at $\pm\infty$ are 1 and 0 respectively. Except for the threshold function, they are also continuous and smooth. The most popular choice is

$$\sigma(x) = 1/(1 + \exp(-x)). \tag{2.3}$$

However, the density proofs do not use all these conditions. For the basic results only continuity or uniform continuity is required, plus in some cases the condition that σ be sigmoidal.

We are interested, then, in approximation by linear combinations of ridge functions. The first papers to establish that one hidden layer is sufficient, i.e. that functions of the form (2.2) are dense, were Cybenko (1989), Hornik *et al.* (1989) and Funahashi (1989). However, simpler and sharper proofs have since superseded this work. An excellent survey of this topic is already available: Light (1992), to which the current author is considerably indebted. Rather than merely repeat the contents of this survey, we will adopt a more synthetic and comparative approach, taking aspects from the methods of the various authors and considering also some work by Mhaskar which postdates the Light survey. Another survey, which deals with some approximation theory issues in neural nets in a more practical way, is Mason and Parks (1992), but this contains little analytical detail. We will also concentrate on the particular practical issues involved for neural networks, but we shall attempt to explain and simplify some of the analytical details which tend to be rather technical in this field. In Sections 2.1 and 2.2 we will give two complete and quite different proofs of the fundamental density result, together with some other interesting sidelights. In the rest of Section 2 we discuss more briefly two other relevant aspects of the interaction of neural computing and approximation theory, namely radial basis networks, and networks using finite length arithmetic.

2.1. Direct approaches to density

Several proofs of the density result start by considering the one-dimensional case and we will also adopt this strategy in this subsection. A direct constructive approximation operator has been devised for this case by Chen, Chen and Liu (1991) (their result is most easily found in Light (1992).) It is of the type known as a *quasi interpolant* which means that it is based on linear combinations of function values. (It does not actually interpolate of course.) The operator is, in fact, similar to the well known Bernstein operator for polynomial approximation. The basic idea is to approximate the function f by piecewise constants: a fairly natural approach in a neural net context where the activation functions are generally thought of as smoothed thresholds. We start by recalling a basic measure of continuity. Let K be a compact set in $\mathbb{R}$. Recall that if $f \in C(K)$, the *modulus of continuity* of f is defined as

$$\omega(f,\delta) = \sup_{\substack{x,y\in K \\ |x-y|<\delta}} |f(x) - f(y)|.$$

Since f is continuous, ω is finite and tends monotonically to zero as $\delta \to 0$. The modulus of continuity encapsulates various ideas of smoothness of f that might be introduced. For example, suppose that f is Lipschitz, i.e. $|f(x) - f(y)| < L|x - y|$ for some real number L and for all $x, y \in K$. Then we have immediately $\omega(f,\delta) \leq L\delta$. Similar estimates apply for α-Lipschitz and for differentiable functions. The modulus of continuity gives a simple estimate of how well f can be approximated by piecewise constants. For simplicity (but without much loss of generality as we can always rescale) we choose K to be the interval [0,1], and let $n \in \mathbb{N}$. We consider the step function $h_n(x)$ which takes the value $f(\nu/n)$ in the interval $\nu/n \leq x =< (\nu+1)/n$. Obviously this gives $\|f - h_n\|_\infty \leq \omega(f, 1/n)$. (Actually we could reduce the constant from 1 to $\frac{1}{2}$ by evaluating at mid-points but this choice simplifies the notation.) It is convenient to write

$$h_n(x) = f(0) + \sum_{\nu=1}^{\mu} \{f(\nu/n) - f((\nu-1)/n)\} \tag{2.4}$$

where μ is the largest integer which does not exceed nx.

Now consider a continuous sigmoidal function such as (2.3). If we replace x by ax for some $a > 1$, we steepen σ in the transitional region around $x = 0$. In fact as $a \to \infty$, $\sigma(ax) \to 0$ or 1, according as $x < 0$ or $x > 0$. In other words σ converges pointwise to a simple threshold function with the value $\sigma(0)$ at 0. Thus $\sigma(ax) - \sigma(a(x-1))$ will approach the unit step function which takes the value 1 on (0,1) and 0 elsewhere. In view of the discontinuities at 0 and 1, this convergence cannot be uniform. But Chen *et al.* (1991) noticed that if one combines the constructions of this and the

previous paragraph to approximate f, one *does* get uniform approximation, since the size of the discontinuity approaches 0 as $n \to \infty$. More specifically, suppose σ is continuous (on $\mathbb{R}$) and sigmoidal. They define A_n to be the smallest positive integer such that

$$|\sigma(x)| \leq n^{-1} \text{ for } x \leq -A_n \quad \text{and} \quad (1-n^{-1}) \leq \sigma(x) \leq (1+n^{-1}) \text{ for } x \geq A_n.$$

Then they define the quasi interpolant g_n as

$$g_n(x) = f(0) + \sum_{\nu=1}^{n} \{f(\nu/n) - f((\nu-1)/n)\}\sigma(A_n(nx-\nu)) \tag{2.5}$$

for $x \in [0,1]$. Observe that this is precisely the approximation obtained by the construction described above: for those values of ν in the summation with $x < \nu$, $\sigma(A_n(nx-\nu))$ is approximately zero. A careful estimate yields the following theorem.

Theorem 2.1 There exists a constant c such that for $f \in C[0,1]$,

$$\|f - g_n\|_\infty \leq c\omega(f, 1/n).$$

(Here the uniform norm is taken on the interval [0,1].) Note that c is independent of f and in fact we may choose $c = 4+2S$ where $S = \sup|\sigma(x)|$ for $x \in \mathbb{R}$, i.e. the norm of σ taken on the whole of $\mathbb{R}$

Proof. We have $\|f-g_n\|_\infty = \|f-h_n+h_n+g_n\|_\infty \leq \|f-h_n\|_\infty + \|h_n-g_n\|_\infty$. We already know $\|f-h_n\|_\infty \leq \omega(f,1/n)$ so only the second term need be considered. Now for any $x \in [0,1]$ with μ defined as in (2.5) we have

$$\begin{aligned} h_n(x) - g_n(x) &= f(0) + \sum_{\nu=1}^{\mu} \{f(\nu/n) - f((\nu-1)/n\}\{1 - \sigma(A_n(nx-\nu))\} \\ &\quad + \sum_{\nu=\mu+1}^{n} \{f(\nu/n) - f((\nu-1)/n)\}\sigma(A_n(nx-\nu)). \end{aligned}$$

Now $\nu \leq \mu - 1$ implies $nx - \nu \geq 1$ so $|1 - \sigma(A_n(A_n - \nu))| \leq 1/n$ by the definition of A_n. Similarly $\nu \geq \mu + 2$ implies $|\sigma(A_n(nx-\nu))| \leq 1/n$. Thus

$$\begin{aligned} |h_n(x) - g_n(x)| &\leq \omega(f,1/n) + |\{f(\mu/n) - f((\mu-1)/n\} \\ &\quad + \{f((\mu+1)/n)\}\sigma(A_n(nx-\mu-1))|. \end{aligned}$$

The second term on the right-hand side is bounded by $2(1+S)\omega(f,1/n)$, which completes the proof. □

We remark in passing that if σ is monotonic then $S = 1$ and c in Theorem 2.1 may be chosen as 6.

Now we need to pass to the n-dimensional case. There are two well known methods of passing from one-dimensional to higher-dimensional approximations: the blending operator and the tensor product. The former method

has not to this author's knowledge, been applied to neural nets at all: the 'infinite interpolation' properties of blending operators seem likely to cause severe problems. However, the tensor product approach offers more hope.

To illustrate both the idea and the problems we will consider briefly the two-dimensional case. Suppose we have two sets of basis functions $\{\phi_1, \ldots, \phi_\mu\}$ and $\{\psi_1, \ldots, \psi_n\}$ where $\phi_i, \psi_j : \mathbb{R} \to \mathbb{R}$ The *tensor product basis* is the set of $\mu \times \nu$ functions

$$\zeta_{i,j}(x, y) = \phi_i(x)\psi_j(y).$$

Sometimes one can construct a two-dimensional approximation using the tensor product basis by applying a one-dimensional approximation operator in each dimension: for example two-dimensional orthogonal expansions can be constructed in this way. In practice the two sets are usually the same type of function (e.g. both polynomials or both trigonometric functions) although μ and ν may of course be different. Now let us consider what happens if we apply this construction to ridge functions. For simplicity we assume that the same function σ is to be used for x and y. So typical one-dimensional ridge functions will be $\sigma(a_i x + c_i)$ and $\sigma(b_j y + d_j)$. The tensor product basis thus consists of functions of the form

$$\sigma(a_i x + c_i)\sigma(b_j y + d_j).$$

In general this does *not* give a two-dimensional ridge function so we will not land up with a neural net approximation of the form (2.2). However, there is one particular choice of σ for which the construction *does* work, namely $\sigma(x) = \exp(x)$. For then we get

$$\begin{aligned} \sigma(a_i x + c_i)\sigma(b_j y + d_j) &= \exp(a_i x + c_i)\exp(b_j y + d_j) \\ &= \exp(a_i x + b_j y + c_i d_j) \\ &= \sigma(a_i x + b_j y + c_i d_j). \end{aligned}$$

This observation has been used by several authors to produce n-dimensional ridge function approximations. The basic idea is to prove the density of the ridge functions for the special case of $\sigma(x) = \exp(x)$ and then to use a one-dimensional result such as Theorem 2.1 to approximate the exponential function by linear combinations of the desired σ. If we are not interested in constructive methods then a simple application of the Stone–Weierstrass theorem (Cheney, 1966, p. 190) will do for the first stage (Diaconis and Shashahani, 1984; see also Hornik *et al.*, 1989). To avoid writing down explicit linear combinations of the form (2.2) all the time, we introduce the following definition: a set of functions is said to be *fundamental* in a space if linear combinations of them are dense in that space.

Theorem 2.2 Let K be a compact set in $\mathbb{R}^n$. Then the set E of functions of the form $\mu(\mathbf{x}) = \exp(\mathbf{a}^T\mathbf{x})$, where $a \in \mathbb{R}^n$, is fundamental in $C(K)$.

Proof. By the Stone–Weierstrass Theorem we need only show that the set forms an algebra and separates points. Suppose $\mathbf{x} \in K$. We first have

$$\exp(\mathbf{a}^T\mathbf{x})\exp(\mathbf{b}^T\mathbf{x}) = \exp(\mathbf{a}^T\mathbf{x} + \mathbf{b}^T\mathbf{x}) = \exp((\mathbf{a}^T + \mathbf{b}^T)\mathbf{x}).$$

The set also contains the function '1': simply choose $\mathbf{a} = 0$. This establishes that E is an algebra. It remains to show that E separates the points of K. So let $\mathbf{x}, \mathbf{y} \in K$ with $\mathbf{x} \neq \mathbf{y}$. Set $\mathbf{a} = (\mathbf{x} - \mathbf{y})$. Then $\mathbf{a}^T(\mathbf{x} - \mathbf{y}) \neq 0$ so $\mathbf{a}^T\mathbf{x} \neq \mathbf{a}^T y$. Thus $\exp(\mathbf{a}^T\mathbf{x}) \neq \exp(\mathbf{a}^T\mathbf{y})$. The proof is complete. □

Before considering more constructive versions of this result let us complete the density proof.

Theorem 2.3 Let K be a compact set in $\mathbb{R}^n$. Then the set F of functions of the form $g(x)$, defined by (2.2) with σ a continuous sigmoidal function, is dense in $C(K)$.

Proof. Let $f \in C(K)$. For any $\epsilon > 0$, there exists (by Theorem 2.2) a finite number m of vectors $\mathbf{a}_i$ such that

$$\left\| f - \sum_{i=1}^{m} \exp(\mathbf{a}_i^T\mathbf{x}) \right\|_\infty < \epsilon/2.$$

Since there are only m scalars $\mathbf{a}_i^T\mathbf{x}$, we may find a finite interval including all of them. Thus there exists a number Γ such that $\exp(\mathbf{a}_i^T\mathbf{x}) = \exp(\Gamma y)$ where $y = (\mathbf{a}_i^T\mathbf{x}/\Gamma) \in [0, 1]$. Then Theorem 2.1 tells us that the function $\exp(\Gamma y)$ can be approximated by linear combinations functions of the form $\sigma(\mathbf{w}_j^T\mathbf{x} + c_j)$ with a uniform error less than $\epsilon/(2m)$, from which the desired result easily follows. □

Sun and Cheney (see e.g. Light (1992) p. 4) have a more sophisticated version of this argument which shows that the elements of the vectors $\mathbf{w}_j$ and the constants c_j may be chosen to be rational numbers (i.e. there is a countable fundamental set). We will not give the details of their result as it complicates the proof significantly. However it is possible to draw the same conclusion by adopting a more constructive approach to Theorem 2.2. Observe first that Γ in the proof of Theorem 2.3 can be chosen to be an integer, and the numbers A_n, n and ν in (2.5) are also integers. The only problem therefore is to show that the vectors $\mathbf{a}$ in Theorem 2.2 can be chosen with rational elements. Moreover, this is also the only part of the argument above that is not constructive. Chen *et al.* (1991) and Mhaskar and Micchelli (1992) get round this by handling the approximation problem of Theorem 2.2 more explicitly. We will not give full analytical details, but indicate the method of attack. Once again, this is based on a tensor product basis. For simplicity, suppose that K is the Cartesian product of closed intervals. Then if a function $f \in C(K)$ is piecewise smooth, it is well

known that f may be expanded in a multivariate Fourier series which will converge uniformly. (If f or K is more complicated, simple expansion is not sufficient, but methods of classical approximation theory may still be used to obtain uniform approximations.) Multiplying out terms one obtains an approximation to f as a linear combination of functions of the form

$$\begin{aligned}&\exp(\mathrm{i}m_1x_1)\exp(\mathrm{i}m_2x_2)\exp(\mathrm{i}m_3x_3)\ldots\exp(\mathrm{i}m_nx_n)\\&\quad=\exp\{\mathrm{i}(m_1x_1+m_2x_2+\ldots+m_nx_n)\},\end{aligned}$$

where $\mathrm{i}^2=-1$ and the $m_j\in\mathbb{N}$. This is precisely of the form required for Theorem 2.2 except for the introduction of complex numbers. Actually the terms occur in complex conjugate pairs so the simplest way to proceed is to use trigonometric functions instead of the exponential function in Theorem 2.2. This will make no difference as far as proof of Theorem 2.3 is concerned.

The method of the previous paragraph also allows the classical Jackson theorems for trigonometric approximation to be used (Cheney, 1966, pp. 139–49) to obtain an estimate of the degree of approximation like that of Theorem 2.1. This question of rate of convergence of approximations is obviously of considerable importance. If f is smooth and we use smooth approximating functions such as (2.3) we might hope to get better convergence than the simple $\mathcal{O}(1/n)$ implied by Theorem 2.1. Apart from a paper by Mhaskar which we consider shortly, very little attention has been given to this issue. Although natural in a neural net context, approximating by piecewise constants as in Theorem 2.1 is a rather odd approach from the viewpoint of classical approximation theory since obviously it means that we cannot possibly do better than $\mathcal{O}(1/n)$.

To get a better degree of approximation a different starting point is required: one such approach will be discussed in Section 2.2. But it is also important to consider which functions we are trying to approximate. To solve the classification problem described at the start of Section 2, approximation by discontinuous functions may seem more natural. Even in analogue applications, it is well known that time optimal controllers may be discontinuous. (See Sagan (1969, pp. 295–97) for a simple example.) Thus, while it is certainly interesting to discuss degree of approximation of smooth functions by smooth networks, it would also be of value to consider degree of approximation to discontinuous functions by nonsmooth or discontinuous networks. Clearly this will require abandonment of the uniform norm and yet the classical L_p norms are not obviously appropriate either. It will be necessary to examine the applications to derive appropriate measures of smoothness and error: these may well be Sobolev or measure theoretic based. In addition we observe that Figure 4 might suggest that two hidden layers rather than one would be preferable. The argument discussing this figure makes use of

geometric ideas of closeness: points which are close together in space can be handled by groups of neurons which deal only with these points. The success of spline approximations suggests that this is a desirable property, and it also makes sense from the point of view of understanding knowledge organization in networks.

Mhaskar (1993) does not solve these problems but he does have some interesting new insights. Already in Mhaskar and Micchelli (1992) the idea of a kth degree sigmoidal function is introduced. A function σ is said to be kth *degree sigmoidal* if

$$\sigma(x)/x^k \to 1 \quad \text{as } x \to \infty$$

and

$$\sigma(x)/x^k \to 0 \quad \text{as } x \to -\infty.$$

The case $k = 0$ recovers the ordinary sigmoidal functions. Functions of this type can be used to approximate a spline of degree k. The idea of Mhaskar and Micchelli was to replace the piecewise constant function (2.4) by a spline of higher degree, thus obtaining a better degree of approximation to smooth f. They then approximate the spline terms by kth degree sigmoidal functions to obtain an approximation of the form (2.2) with a better degree of approximation to f if $k > 0$. (The analytical details are conveniently found in Light (1992, pp. 28–30).) Of course σ is no longer the conventional sigmoidal function normally used for neural nets. In the later article Mhaskar uses these ideas in new way. First he deals with the multivariate case directly, approximating the multivariate f by a tensor product spline. He then considers the problem of approximating f by a neural network with a fixed number of neurons but arranged in more than one layer. Both the cases $k = 0$ and $k > 0$ are considered; the details are slightly different for the two cases although the basic idea is the same. Multilayer networks involve compositions of linear combinations of sigmoidal functions. Mhaskar employs the fact that the polynomial terms in the splines can themselves be decomposed as compositions of linear functions which can be approximated by the sigmoidal functions. Several interesting results are obtained on the degree of approximation obtainable by this method when $k > 0$. Unfortunately the technical details of Mhaskar's arguments are quite complicated, but there is no doubt that the article will repay close study. But of course, k sigmoidal functions are not the functions actually used in neural nets.

2.2. Dual space and convolution methods of approximation

Readers familiar with approximation theory will be aware that as well as the direct constructive approach to density, it is also possible to achieve such results by methods based on integrals, generally based either on dual space

arguments or convolutions. These methods can be used to address the question of density of networks, the most recent work here being the convolution technique of Xu *et al.* (1991) who obtain constructive approximations this way. Although their work requires rather more stringent conditions on σ than the methods of the previous section, it does have certain advantages. In particular, being based on the use of quadrature formulae, it would seem to present the possibility of a line of attack for estimating the degree of approximation to smooth f.

Most of this section will therefore be devoted to convolutions, but first in the interests of completeness we consider dual space methods. This topic is of at least historical interest, since Cybenko's original proof was of this type (Cybenko, 1989). Since this approach may be rather mystifying to approximation-theory nonspecialists, we will first describe the fundamental idea of the method. Consider a normed vector space X over $\mathbb{R}$ The (bounded) *dual space* of X, denoted by X', is the space of all bounded linear functionals on X. (A linear functional is a linear mapping from X to $\mathbb{R}$) It has a natural norm defined by

$$\|l\| = \sup_{\substack{\mathbf{x}\in X \\ \|\mathbf{x}\|=1}} |l(\mathbf{x})|. \tag{2.6}$$

X' is always a Banach space, even if X is not. (Readers not familiar with this construction at all are advised to consult a suitable textbook such as Kreyszig (1978, pp. 119–25).)

Now let V be a subspace of X. We wish to know whether V is dense in X. The relevance of the dual space is shown by the following theorem.

Theorem 2.4 V is dense in X if and only if the only linear functional $l \in X'$ for which $l(\mathbf{v}) = 0$ for all $v \in V$ is the trivial one $l(\mathbf{x}) \equiv 0$.

Proof. Suppose first that V is dense in X. Suppose also that l is a linear functional l such that $l(v) = 0$ for $\mathbf{v} \in V$. Let $\mathbf{x} \in X$. For any $\epsilon > 0$ we have $\mathbf{v} \in V$ with $\|\mathbf{x} - \mathbf{v}\| < \epsilon$. Then $|l(\mathbf{x})| = |l(\mathbf{x}) - l(\mathbf{v})| = |l(\mathbf{x} - \mathbf{v})| \leq \|l\|\|\mathbf{x} - \mathbf{v}\| < \|l\|\epsilon$. Since this is true for any $\epsilon > 0$, we must have $l(\mathbf{x}) = 0$. This establishes the 'only if' part of the theorem.

Now suppose that V is *not* dense in X. Then there is a $\mathbf{x} \in X$ and a number $\delta > 0$ such that $\|\mathbf{x} - \mathbf{v}\| > \delta$ for all $\mathbf{v}$ in V. Let W be the space spanned by $\mathbf{x}$ and the space V, i.e. the set of all linear combinations $\alpha\mathbf{x} + \mathbf{v}$, where $\alpha \in \mathbb{R}$ and $\mathbf{v} \in V$. Note that the number α here is unique, for if $\alpha_1 x + \mathbf{v}_1 = \alpha_2\mathbf{x} + \mathbf{v}_2$ we have $(\alpha_1 - \alpha_2)\mathbf{x} = \mathbf{v}_2 - \mathbf{v}_2$, whence we must have $\alpha_1 = \alpha_2$ since $\mathbf{x} \notin V$. Thus we can define the following linear functional on W : $l(\alpha\mathbf{x} + \mathbf{v}) = \alpha$. Note that $l(\mathbf{x}) = 1$ and $l(\mathbf{v}) = 0$ for all $\mathbf{v} \in V$. Now if $\mathbf{w} = \alpha\mathbf{x} + \mathbf{v}$ with $\alpha \neq 0$,

$$\|\alpha\mathbf{x} + \mathbf{v}\| = |\alpha|\|\mathbf{x} + \alpha^{-1}\mathbf{v}\| \geq |l(w)|\delta$$

so

$$|l(w)| \leq \|\mathbf{w}\|/\delta.$$

On the other hand if $\alpha = 0$, $|l(w)| = 0$ so the inequality above holds trivially. This shows that l is a nontrivial bounded linear functional on W. By the Hahn–Banach Theorem (Kreyszig, 1978 p. 214) l may be extended to a bounded linear functional on the whole of X. This completes the proof. □

Thus if we want to establish density of V in X we need only show that any linear functional which annihilates V is, in fact, the zero functional. This may at first sight seem a harder task than the original one. However for most function spaces used in practice it is possible to get a concrete representation of X' and its norm. This makes the task tractable. In fact it may be shown (Kreyszig, 1978, p. 227) that for the case of $X = C[a, b]$, any linear functional may be written

$$l(f) = \int_a^b f(x)\,\mathrm{d}w(x), \tag{2.7}$$

where w is a function of bounded variation. (Readers unfamiliar with the interpretation of this integral are again referred to Kreyszig (1978, p. 226), but it is convenient to use the Lebesgue integral here rather than the Riemann integral employed by Kreyszig.) So to establish the density of one-dimensional sigmoidal functions (compare Theorem 2.1) we need only show that if the integral (2.7) vanishes whenever f is a sigmoidal function, then necessarily w is constant. This is fairly straightforward. We have

$$0 = \int_a^b \sigma(kx + l)\,\mathrm{d}w(x) \tag{2.8}$$

for all $k, l \in \mathbb{Z}$. Once again we adopt the basic idea of making k large enough so that the integrand looks like a step function. For any $p, q \in \mathbb{Z}$ with $p/q \in [a, b]$, define

$$r(x) = \begin{cases} 0 & a < p/q \\ \sigma(l) & x = p/q \\ 1 & p/q < x \leq b. \end{cases}$$

Now consider the expression $\sigma(nq(t - p/q))$. As $n \to \infty$, this expression converges pointwise to r on $[a, b]$. By the Lebesgue Dominated Convergence Theorem we conclude that

$$0 = \int_a^b r(x)\,\mathrm{d}w(x) = \int_{(p/q)^+}^b \mathrm{d}w(x) + \sigma(l)(w(p/q^+) - w(p/q^-)).$$

The final term denotes the jump in w at the point p/q. Notice that the integral term does not depend on l so we first let $l \to -\infty$. By the definition of a sigmoidal function, $\sigma(l) \to 0$. We conclude that the integral term is

zero. By subtraction we may deduce that

$$0 = \int_t^s \mathrm{d}w(x) \tag{2.9}$$

whenever s, t are rational and (by the Lebesgue Dominated Convergence Theorem using sequences of rational numbers to converge monotonically to the required end points) we deduce that in fact (2.9) holds for any $t, s \in [a, b]$. But this integral is precisely $w(s) - w(t)$, showing that w is, in fact, constant as required.

Although this result captures the main significance of Theorem 2.1, it is not as powerful since it does not yield the $\omega(f, 1/n)$ estimate, and it is not clear how such estimates could be obtained by this approach. So we will look instead at the use of convolutions.

The idea of the convolution method is to construct a kernel based on the sigmoidal functions which can be used to approximate the reproducing property of the Dirac generalized function. The convolution itself and the kernel are, in turn, approximated by quadrature formulae thus yielding the required approximation. The material discussed here is essentially that of Xu *et al.* (1991) as expounded in that article and in Light (1992). However, our development will differ in one or two respects.

We first require the basic reproducing property. A problem presents itself immediately in that the standard Theorem 2.5 below, as quoted by Xu *et al.* (Stein and Weiss, 1971, p. 10) requires the function f to be uniformly continuous on the whole of $\mathbb{R}^n$. In the approximation-theory context, it is more natural to consider a compact subset K. We therefore need to extend f from K to $\mathbb{R}^n$ in a suitable way. Xu *et al.* take K to be $[-\nu, \nu]$ and do not give the extension explicitly, but clearly such an extension is possible. We use $\mathrm{d}V(\mathbf{x})$ to denote the volume element $\mathrm{d}x_1\,\mathrm{d}x_2 \ldots \mathrm{d}x_n$. (Both Stein and Weiss, and Xu *et al.* use just $\mathrm{d}x$, but we prefer to emphasize that a volume integral is involved.) To avoid confusion here we also use $\|\cdot\|$ to denote the ordinary Euclidean norm on $\mathbb{R}^n$. Norms on the function spaces will be subscripted 1 or ∞ as appropriate. We also need to extend the notion of modulus of continuity to $\mathbb{R}^n$: this is simply defined to be

$$\omega(f, \delta) = \sup_{\substack{\mathbf{x},\mathbf{y}\in K \\ \|\mathbf{x}-\mathbf{y}\|<\delta}} |f(\mathbf{x}) - f(\mathbf{y})|.$$

Here $K \subset \mathbb{R}^n$ does not need to be compact, but we do need f to be uniformly continuous on K. We will give a proof of the required convolution result here, and add a couple of bounds that in some cases might enable results to be sharpened. Although we will not discuss these in detail in the rest of the article, it seems worthwhile to give them in order to stimulate further research on sharp estimates.

Theorem 2.5 Let f be bounded and uniformly continuous on $\mathbb{R}^n$ and let

$g \in L_1(\mathbb{R}^n)$ with

$$\int_{\mathbb{R}^n} g(\mathbf{x})\,\mathrm{d}V(\mathbf{x}) = 1.$$

Define $g_m(\mathbf{x}) = m^n g(m\mathbf{x})$. Then

(a) $f * g_m(x)$ converges uniformly to f as $m \to \infty$.

(b) For any $R > 0$,

$$\|f * g_m - f\|_\infty \leq \omega(f, 2R/m)\|g\|_1 + 2\|f\|_\infty \int_{\|s\|>R} |g(\mathbf{s})|\,\mathrm{d}V(\mathbf{s}),$$

where $\|.\|_\infty$ is taken over the whole of $\mathbb{R}^n$.

(c) As an alternative to condition (b), suppose that f is Lipschitz with constant Λ and that

$$M = \int_{\mathbb{R}^n} \|\mathbf{x}\||g(\mathbf{x})|\,\mathrm{d}V(\mathbf{x}) < \infty.$$

Then $\|f * g_m - f\|_\infty \leq M\Lambda/m$.

Proof. First observe that

$$\begin{aligned}\int_{\mathbb{R}^n} g_m(\mathbf{x})\,\mathrm{d}V(\mathbf{x}) &= \int_{\mathbb{R}^n} m^n g(m\mathbf{x})\,\mathrm{d}V(\mathbf{x}) = \int_{\mathbb{R}^n} g(m\mathbf{x})\,\mathrm{d}V(m\mathbf{x}) \\ &= 1 \quad (\text{setting } \mathbf{y} = m\mathbf{x}).\end{aligned}$$

Hence

$$(f * g_m)(\mathbf{x}) - f(\mathbf{x}) = \int_{\mathbb{R}^n} (f(\mathbf{x}-\mathbf{t}) - f(\mathbf{x}))g_m(\mathbf{t})\,\mathrm{d}V(\mathbf{t})$$

so

$$\begin{aligned}|(f * g_m)(\mathbf{x}) - f(\mathbf{x})| &\leq \int_{\mathbb{R}^n} |f(\mathbf{x}-\mathbf{t}) - f(\mathbf{x})||g_m(\mathbf{t})|\,\mathrm{d}V(\mathbf{t}) \\ &= \int_{\mathbb{R}^n} |f(\mathbf{x}-\mathbf{t}) - f(\mathbf{x})||g(m\mathbf{t})|m^n\,\mathrm{d}V(\mathbf{t}) \\ &= \int_{\mathbb{R}^n} |f(\mathbf{x}-\mathbf{s}/m) - f(\mathbf{x})||g(\mathbf{s})|\,\mathrm{d}V(\mathbf{s}) \quad (2.10) \\ &\quad \text{where } \mathbf{s} = m\mathbf{t} \\ &\leq \int_{\mathbb{R}^n} \omega(f, \|\mathbf{s}\|/m)|g(\mathbf{s})|\,\mathrm{d}V(\mathbf{s}).\end{aligned}$$

Clearly $\omega(f, \|\mathbf{s}\|/m)$ is an integrable function of f and converges monotonically pointwise to zero as $m \to \infty$. Hence the integral on the right goes to zero by the Monotone Convergence Theorem. This establishes (a).

To get (b) we simply split the integration in (2.10) into the two regions $\|\mathbf{s}\| \leq R$ and $\|\mathbf{s}\| > R$ and bound each term.

Part (c) is also straightforward: simply write $|f(\mathbf{x} - \mathbf{s}/m) - f(\mathbf{x})| < \Lambda\|\mathbf{s}\|/m$ in (2.10). □

The bound given in part (b) of the theorem does not tend to zero as $m \to \infty$ unless g has compact support, in which case we can choose R so that the second term is zero. Since we are going to construct g as an integral of our sigmoidal functions, compact support will be hard to achieve. On the other hand, we shall often find that $g(\mathbf{x})$ goes to zero very rapidly as $\|\mathbf{x}\| \to \infty$, so the bound might well be useful for finite m. Moreover the conditions of part (c) are likely to hold in many cases of practical interest. But this will still not yield any better estimates than those we already have from Section 2.1. Nevertheless, this approach does seem likely to reward further work with sharper estimates.

Equipped with this basic tool on uniform convergence of convolutions, let us now consider the work Xu *et al.* They construct a convolution kernel g as

$$g(\mathbf{x}) = \alpha_{n-1}^{-1} \int_{S^{n-1}} \phi(\mathbf{x}^T\mathbf{u})\, \mathrm{d}S^{n-1}(\mathbf{u}), \tag{2.11}$$

where S^{n-1} is the unit sphere in $\mathbb{R}^n$ (i.e. the set $\{\mathbf{u} \in \mathbb{R}^n \mid \|\mathbf{u}\| = 1\}$ and α_{n-1} is the 'surface area' obtained as the value of the integral with $\phi \equiv 1$. They work on a suitable compact set K in $\mathbb{R}^n$. Their choice is actually the n-dimensional interval $K = [-a, a]^n$ for some real a: the actual choice makes little difference in the following proof. We assume that $f \in C(K)$ and, so that we can apply Theorem 2.5, extended f continuously to $\mathbb{R}^n$ in such a way that $f(\mathbf{x}) = 0$ for $\mathbf{x} \in 2K$ where for any $t > 0$,

$$tK = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x}/t \in K\}.$$

Since f is then continuous on the bounded set $2K$ we conclude that f is uniformly continuous and bounded on $\mathbb{R}^n$. The fundamental result is:

Theorem 2.6 Let $K = [-a, a]^n$ for some real a, and let $\phi \in C(\mathbb{R})$ be uniformly continuous. Suppose g is defined by (2.11). If (i) $g \in L_1(\mathbb{R}^n)$ and (ii)

$$\int_{\mathbb{R}^n} g(\mathbf{x})\, \mathrm{d}V(\mathbf{x}) \neq 0,$$

then the set of functions of the form $\phi(\mathbf{x}^T\mathbf{a} + c)$, $\mathbf{a} \in \mathbb{R}^n$, $c \in \mathbb{R}$, is fundamental in $C(K)$.

Proof. Let $f \in C(K)$. We extend f to a bounded uniformly continuous function on $\mathbb{R}^n$ as described in the paragraph before the theorem. Rescaling ϕ if necessary, we can assume that the integral of g is 1. Now suppose we are given $\epsilon > 0$. In view of Theorem 2.5 we may choose m such that

$$\|f * g - f\|_\infty \leq \epsilon/3,$$

where here we restrict $\|\cdot\|_\infty$ to K. (Note that there would appear to be a minor error in the the proof of this theorem in both Xu *et al.* (1991, p. 12)

and Light (1992, p. 18). The bound in terms of $\omega(f, m^{-1})$ is not, in fact, correct; compare Theorem 2.5(b). Nevertheless the bound as given above is valid.)

The next step is to approximate the convolution by a quadrature formula. We have

$$(f * g_m)(\mathbf{x}) = \int_{2K} g_m(\mathbf{x} - \mathbf{y}) f(\mathbf{y})\, \mathrm{d}V(\mathbf{y}) = \int_{2mK} g(m\mathbf{x} - \mathbf{z}) f(\mathbf{z}/m)\, \mathrm{d}V(\mathbf{z}). \tag{2.12}$$

It is a simple consequence of (2.11) that g is continuous: indeed $\omega(g, \delta) \leq \omega(\phi, \delta)$. Since the integrand is continuous and $2mK$ is compact we can approximate it by a quadrature formula for any particular value of $\mathbf{x}$. But we need a bound uniform over $\mathbf{x}$. For the present we will just use a simple Riemann sum approximation as do Xu *et al.* (However to get sharper estimates it will prove necessary to give closer attention to the approximation of (2.12).) For any $\delta > 0$, let P be a partition of $2mK$ into a finite disjoint family of Borel sets, each of diameter at most δ. For each $A \in P$, choose $\mathbf{z}_A \in A$ and define

$$b_A = \int_A f(\mathbf{z}/m)\, \mathrm{d}V(\mathbf{z}).$$

So

$$\begin{aligned}
&\left| \int_{2mK} g(m\mathbf{x} - \mathbf{z}) f(\mathbf{z}/m)\, \mathrm{d}V(\mathbf{z}) - \sum_{A \in P} b_A g(m\mathbf{x} - A) \right| \\
&\quad \leq \sum_{A \in P} \int_A |g(m\mathbf{x} - \mathbf{z}) - g(m\mathbf{x} - \mathbf{z}_A)||f(\mathbf{z}/m)|\, \mathrm{d}V(\mathbf{z}) \\
&\quad \leq \omega(g, \delta) \sum_{A \in P} \int_A f(\mathbf{z}/m)|\, \mathrm{d}V(\mathbf{z}) \\
&\quad = m^n \omega(g, \delta) \int_{2K} |f(\mathbf{y})|\, \mathrm{d}V(\mathbf{y}).
\end{aligned}$$

Hence we can choose δ and P so that this error is less than $\epsilon/3$.

Now we apply a similar argument to (2.11). For any $\theta > 0$, let Q be a partition of S^{n-1} into a finite disjoint collection of Borel sets of diameter at most θ. For any $B \in Q$, Set

$$c_B = \alpha_{n-1}^{-1} \int_B \mathrm{d}S^{n-1}(\mathbf{u})$$

and choose $\mathbf{u}_B \in B$. We get in a similar fashion to the argument above

$$|g(m\mathbf{x} - \mathbf{z}_A) - \sum_{B \in Q} c_B \phi((m\mathbf{x} - \mathbf{z}_A)^T \mathbf{u}_B)| \leq \omega(\phi, \|m\mathbf{x} - \mathbf{z}_A\|\theta).$$

But $\mathbf{z}_A \in 2mK$ so $\mathbf{z}_A/m \in 2K$. Hence for any $\mathbf{x} \in K, \|m\mathbf{x} - \mathbf{z}_A\| = m\|\mathbf{x} - \mathbf{z}_A/m\| \leq 3R$, where R is the diameter of K. Thus we can choose

θ and B so that the right-hand side of the inequality above is less than $\|f\|_1\epsilon/3$, whence

$$\left|\sum_{A\in P} b_A\Big(g(m\mathbf{x}-\mathbf{z}_A)-\sum_{B\in Q} c_B\phi((m\mathbf{x}-\mathbf{z}_A)^T\mathbf{u}_B)\Big)\right|\le\epsilon/3.$$

Finally, putting these three approximations together, we find that, for any $\mathbf{x}\in K$,

$$\left|f(\mathbf{x})-\sum_{A\in P}\sum_{B\in Q} b_Ac_B\phi(m\mathbf{x}^T\mathbf{u}_B-\mathbf{z}_A^T\mathbf{u}_B\right|\le\epsilon.\quad\square$$

Thus, to establish that the functions $\phi(\mathbf{x}^T\mathbf{a}+c)$, $\mathbf{a}\in\mathbb{R}^n$, $c\in\mathbb{R}$ are fundamental, we have two points to check: first that g defined by (2.11) is in $L_1(\mathbb{R}^n)$; and second that it has a nonzero integral. We shall find that we cannot simply take $\phi=\sigma$ where σ is our required sigmoidal function; we will need to take a linear combination of σ terms.

The first question we need to consider is how fast g must go to zero to be in $L_1(\mathbb{R}^n)$. We need to do nothing more sophisticated than to bound $g(\mathbf{x})$ by a suitable power of $r=\|x\|$ when r is large. The next lemma tells us what power is required.

Lemma 2.7 With $r=\|\mathbf{x}\|,\mathbf{x}\in\mathbb{R}^n$, and $q,R\in\mathbb{R}$ with $R>0$, we have

$$\int_{\|\mathbf{x}\|\ge R} r^{-q}\,\mathrm{d}V(\mathbf{x})<\infty$$

if and only if $q>n$.

Proof. Denoting the sphere of radius r by S_r^{n-1}, we have for $\rho>R$

$$\int_{\rho\ge\|\mathbf{x}\|\ge R} r^{-q}\,\mathrm{d}V(\mathbf{x})=\int_R^\rho r^{-q}\int_{S_r^{n-1}}\mathrm{d}S_r^{n-1}\,\mathrm{d}r.$$

But since S_r^{n-1} is an $(n-1)$-dimensional manifold,

$$\int_{S_r^{n-1}}\mathrm{d}S_r^{n-1}=r^{n-1}\alpha_{n-1}.$$

(Recall α_{n-1} is the area of the sphere of radius 1: see (2.11).) Thus

$$\int_{\|\mathbf{x}\|\ge R}^{\rho} r^{-q}\,\mathrm{d}V(\mathbf{x})=\alpha_{n-1}\int_R^\rho r^{-q+n-1}\,D\quad=\alpha_{n-1}\left[\frac{r^{n-q}}{n-q}\right]_R^\rho\quad\text{for } q\ne n.$$

Thus the limit as $\rho\to\infty$ exists only if $q>n$. The case $q=n$ yields a logarithmic integral which also goes to infinity. $\square$

Thus to show that $g\in L_1(\mathbb{R}^n)$, it is sufficient to show that $g(\mathbf{x})=o(\|\mathbf{x}\|^{-n})$ as $\|\mathbf{x}\|\to\infty$. To do this Xu *et al.* find an alternative form of (2.11).

Lemma 2.8 Let g be defined by (2.11). Then $g(\mathbf{x}) = g_0(\phi, r)$ where $r = \|\mathbf{x}\|$ and

$$g_0(\phi, r) = \frac{\alpha_{n-2}}{\alpha_{n-1}} \int_{-1}^{1} \phi(rs)(1-s^2)^{(n-3)/2}\,\mathrm{d}s, \tag{2.13}$$

$$= \frac{\alpha_{n-2}}{\alpha_{n-1}} \int_{-r}^{r} r^{2-n}\phi(t)(r^2-t^2)^{(n-3)/2}\,\mathrm{d}t, \quad r \neq 0. \tag{2.14}$$

(A function such as g which depends only on r is said to be *radial.*)

Proof. We may assume $\|\mathbf{x}\| \neq 0$: this point can be 'filled in' as a limiting case since both (2.11) and (2.13) depend continuously on $\mathbf{x}$. So we may choose a coordinate system with its pole in the direction of $\mathbf{x}$. Let $\mathbf{w}$ be a unit vector in the direction of $\mathbf{x}$, whence $\mathbf{w} = \mathbf{x}/\mathbf{r}$. Then any point $\mathbf{u} \in S^{n-1}$ can be expressed as $\mathbf{u} = \mathbf{w}\cos\theta + \mathbf{v}$, where $\cos\theta = \mathbf{u}^T\mathbf{x}/r$, and $\mathbf{v}$ is a unit vector perpendicular to $\mathbf{x}$ with $\|\mathbf{v}\| = \sin\theta$. Moreover we cover the whole of S^{n-1} as θ varies from 0 to π and v takes all directions orthogonal to $\mathbf{x}$. So

$$g(\mathbf{x}) = \alpha_{n-1}^{-1} \int_{S^{n-1}} \phi(\mathbf{x}^T\mathbf{u})\,\mathrm{d}S^{n-1}(\mathbf{u})$$

$$= \alpha_{n-1}^{-1} \int_0^{\pi} \phi(r\cos\theta) \int_{S^{n-2}_{\sin\theta}} \mathrm{d}S^{n-2}_{\sin\theta}(\mathbf{v})\,\mathrm{d}\theta$$

where, as in the proof of Lemma 2.7, $S^{n-2}_{\sin\theta}$ denotes the $(n-2)$-dimensional sphere of radius $\sin\theta$. The inner integral is thus $\alpha_{n-2}\sin^{n-2}\theta$. This establishes that g is indeed a radial function and we can define g_0. Moreover, we have

$$g_0(\phi, r) = \frac{\alpha_{n-2}}{\alpha_{n-1}} \int_0^{\pi} \phi(r\cos\theta)\sin^{n-3}\theta\sin\theta\,\mathrm{d}\theta.$$

Now put $s = \cos\theta$, so $\mathrm{d}s = -\sin\theta\,\mathrm{d}\theta$ and $\sin^{n-3}\theta = (1-s^2)^{(n-3)/2}$ as required. (2.14) is obtained by substituting $t = rs$. □

Now, how do we choose ϕ to give g in $L_1(\mathbb{R}^n)$? The essential requirement is that $\phi(t)$ goes to zero quickly enough at $\pm\infty$. Let us now consider sigmoidal functions specifically. We note from (2.13) that g vanishes if ϕ is odd, so we might as well choose choose ϕ even, although this is not essential. Sigmoidal functions σ tend to 1 at $+\infty$, so let us first define

$$\psi(t) = \sigma(1+t) + \sigma(1-t) - 1. \tag{2.15}$$

Note that ψ is even and goes to zero at $\pm\infty$. Since we expect our functions σ to approximate step functions, it is reasonable to suppose that ψ goes to zero quickly. More specifically, let us suppose that σ is continuous and is such that

$$|\psi(t)| \leq K|t|^{-q} \quad q > n-2 \tag{2.16}$$

for some real K. Note that if σ is the usual choice (2.3), this condition

actually holds for *any* $q > 0$ as ψ goes to zero exponentially. The next step in the argument is to expand the kernel in (2.14). Write $\lambda = (n-3)/2$. From now on we will assume $n > 2$: $n = 2$ requires special treatment (Xu *et al.*, 1991, p. 10), but we will not bother with this here. Let us first consider the case n odd, so that λ is a nonnegative integer. Hence $(r^2 - t^2)^\lambda$ is simply a polynomial in r and t, and the integral (2.14) may be taken termwise. We get

$$g_0(\phi, r) = r^{2-n} \sum_{j=0}^{\lambda} \beta_j(r) r^{2\lambda - 2j} \tag{2.17}$$

where

$$\beta_j(r) = \alpha_{n-2}\alpha_{n-1}^{-1} {}^{\lambda}C_j \int_{-r}^{r} \phi(t) t^{2j} \, \mathrm{d}t. \tag{2.18}$$

(Here ${}^{\lambda}C_j$ is the usual binomial coefficient.) The condition (2.16) means that all the β_j converge as $r \to \infty$. Hence:

Lemma 2.9 Let n be odd and ψ satisfy (2.16) and g be defined by (2.11) with $\psi = \phi$. A necessary and sufficient condition for g to be in $L_1(\mathbb{R}^n)$ is that

$$\int_0^\infty \psi(t) t^{2j} \, \mathrm{d}t = 0, \quad j = 0, \ldots, (n-3)/2. \tag{2.19}$$

Proof. If (2.19) fails for some j, we see from (2.17) that as $r \to \infty$, $g_0(\phi, r)$ would behave like r^{-p}, where

$$p = n - 2 - 2\lambda + 2j \leq n - 2 < n$$

(compare Lemma 2.7). To get the sufficiency we note that (2.17) and (2.19) together imply that

$$g_0(\phi, r) = -r^{2-n} \sum_{j=0}^{\lambda} \gamma_j(r) r^{2\lambda - 2j} \tag{2.20}$$

with

$$\begin{aligned} \gamma_j(r) &= \alpha_{n-2}\alpha_{n-1}^{-1} {}^{\lambda}C_j \int_{|t|>r} \psi(t) t^{2j} \, \mathrm{d}t \\ &\leq \alpha_{n-2}\alpha_{n-1}^{-1} {}^{\lambda}Cj \int_{|t|>r} K|t|^{-q} t^{2j} \, \mathrm{d}t \\ &= 2K\alpha_{n-2}\alpha_{n-1}^{-1} {}^{\lambda}C_j \int_r^\infty t^{2j-q} \, \mathrm{d}t \\ &= -2K\alpha_{n-2}\alpha_{n-1}^{-1} {}^{\lambda}C_j r^{2j-q+1} \end{aligned}$$

since by hypothesis $q > n - 2 > 2j$. Substituting this bound into (2.20) we

find that g_0 goes to zero at least as fast as r to the power

$$(2 - n + 2\lambda - 2j + 2j - q + 1) = (2 - n + n - 3 - q + 1) = -q, \quad \text{and } q > n.$$

(At at least the same rate as ψ, in fact.) Thus $g(\|x\|)$ goes to zero sufficiently fast. □

At first sight it appears from from (2.17), (2.18) and (2.19) that there is little hope of finding a nonzero g. But Xu *et al.* observed that this is not the case. First we will see that the moment condition (2.19) poses no serious difficulty. If we replace $\phi(t)$ by $\phi(pt)$ where $p > 1$, we have

$$\int_0^\infty \phi(pt)t^{2j}\,\mathrm{d}t = p^{-(2j+1)} \int_0^\infty \phi(\mathbf{s})s^{2j}\,\mathrm{d}s \quad \text{by the substitution } s = pt. \tag{2.21}$$

If we have ψ (not identically zero) satisfying (2.16) we may choose (say) $p = 2$ and define $\psi_0(t) = \psi(t)$, and $\psi_j(t) = \psi_{j-1}(t) - 2^{2j+1}\psi_{j-1}(2t)$, $j = 0, \ldots, \lambda$. Observe that ψ_j will still satisfy (2.16) with the same order q but with the previous K replaced at each stage by $(1 + 2^{2j+1-q})K$. Also (2.21) means that the moment condition of (2.19) is satisfied for powers of t up to j: thus $\phi(t) = \psi_\lambda(t)$ will satisfy (2.19) for all the required values of j. (The reader might be concerned that ψ_λ could vanish identically. However this will turn out to be impossible in the context we are going to use the result.)

It remains to show that the resulting g cannot have zero integral. Xu *et al.* prove the following elegant result.

Lemma 2.10 Let n be odd and ψ satisfy the conditions of Lemma 2.10 including (2.19). Suppose also ψ is even. Then

$$\int_{\mathbb{R}^n} g(\mathbf{x})\,\mathrm{d}V(\mathbf{x}) = -2\alpha_{n-2}\tau_n \int_0^\infty \psi(t)t^{n-1}\,\mathrm{d}t$$

where

$$\tau_n = \int_0^1 r(1 - r^2)^{(n-3)/2}\,\mathrm{d}r > 0.$$

Proof.

$$\begin{aligned}
\int_{\mathbb{R}^n} g(\mathbf{x})\,\mathrm{d}V(\mathbf{x}) &= \alpha_{n-1} \int_0^\infty r^{n-1} g_0(r)\,\mathrm{d}r, \\
&\quad \text{as in the proof of Lemma 2.7} \\
&\quad \text{with } r^{-q} \text{ replaced by } g_0(r). \\
&= 2\alpha_{n-2} \int_0^\infty r \int_0^r \psi(t)(r^2 - t^2)^{(n-3)/2}\,\mathrm{d}t\,\mathrm{d}r, \\
&\quad \text{by (2.14) since } \psi \text{ is even} \\
&= -2\alpha_{n-2} \int_0^\infty r \int_r^\infty \psi(t)(r^2 - t^2)^{(n-3)/2}\,\mathrm{d}t\,\mathrm{d}r \\
&\quad \text{by (2.19)}
\end{aligned}$$

$$= -2\alpha_{n-2}\int_0^\infty \psi(t)\int_0^t r(r^2-t^2)^{(n-3)/2}\,dr\,dt,$$

by Fubini's theorem.

But the inner integral is a constant multiplied by t^{n-1}, and by putting $t=1$ we see that the value of the constant is τ_n. τ_n is certainly strictly positive, as its integrand is positive except at 0 and 1. □

Note that it is no real restriction that ψ be even; we can make it so as in (2.15). Putting all this together we arrive at the following theorem.

Theorem 2.11 Let $\sigma \in C(\mathbb{R})$ and ψ defined by (2.15). Suppose that n is odd and K is defined as for Theorem 2.5. Suppose also that ψ satisfies (2.16), and that

$$\int_0^\infty \psi(t)t^{n-1}\,dt \neq 0.$$

Then the set of functions $\sigma(\mathbf{x}^T\mathbf{a}+c)$, $\mathbf{a}\in\mathbb{R}^n$, $c\in\mathbb{R}$, is fundamental in $C(K)$. In particular, this is true if σ is defined by (2.3).

Proof. This result is basically just an application of Theorems 2.6 and 2.11, but we do have to worry about the moment condition (2.19). Instead of ψ in (2.19) we must use ψ_λ as defined in the paragraph after (2.21). By a similar argument to (2.21) we find that for each j in this definition,

$$\int_0^\infty \psi_j(t)t^{n-1}\,dt = (1-2^{2j+1-n})\int_0^\infty \psi_{j-1}(t)\,dt.$$

Since $j \le (n-3)/2$, so the power cannot be zero, we find that the integral vanishes for $j=\lambda$ if and only if it vanishes for $j=0$. But by definition $\psi_0=\psi$.

If σ is as defined in (2.3), then a routine calculation shows $\psi(t) = (1-e^{-2})/(1+e^{-2}+2e^{-1}\cosh(t)) > 0$ for all t. So the integral in the statement of the theorem does not vanish. □

We remark also that the restriction $q>n$ in condition (2.16) can also be relaxed if ψ is sufficiently well behaved at ∞. More specifically, suppose

$$\psi(t) \simeq Ht^{-q} \quad \text{when } q \text{ is large.}$$

Then a binomial expansion shows that

$$\psi(t+1)-\psi(t-1) \simeq -Hqt^{-(q+1)}.$$

(In fact (2.15) gives one higher order for ψ than σ.) We may repeat this process until the exponent is greater than n. However, we might then have difficulty in showing that the integral in Theorem 2.11 does not vanish!

The case n even is rather less satisfactory. Lemma 2.9 still holds with in this case $j=0,\dots,(n-2)/2$ in (2.19). (Note the increased upper limit.)

The argument is essentially the same, although a little more care is needed since the expansion of the kernel is no longer finite in (2.17): we have to replace the upper limit of the sum by ∞. Then we need to justify the termwise integration: details are given in Light (1992, p. 16). However, the increased power of t that must be annihilated causes a problem. The first nonvanishing power of t is n. Thus the first nonvanishing power of r is $(2-n)+(-n-3+n)=-(n+1)$. To obtain the integral of g itself we must multiply by r^{n-1} and integrate (compare the proof of Lemma 2.10). The leading power to be integrated is therefore r^{-2} which integrates to a $1/r$ term. This suggests that the integral of g will in fact vanish. Xu *et al.* devote several pages of analysis to justifying this formally (a process which unfortunately tends to hide the fact that it is essentially a power counting argument). Since the integral of g vanishes, we cannot apply Theorem 2.5 directly to the case n even. However this is overcome by averaging in one higher dimension. Specifically, define

$$h(\mathbf{x}) = \alpha_n^{-1} \int_{S^n} \phi(\mathbf{x}'^T \mathbf{u}) \, \mathrm{d}S^n(\mathbf{u}),$$

where for $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, $\mathbf{x}' = (x_1, x_2, \ldots, x_n, 0) \in \mathbb{R}^{n+1}$. Using similar arguments to those above, we can show that h is a suitable kernel.

Now we have not actually used the fact that σ is sigmoidal, only that ψ defined by (2.15) be uniformly continuous and satisfy (2.16). Xu *et al.* also use a slightly weaker condition than (2.16), although it amounts to the same thing for practical σs. But apart from these minor considerations Theorem 2.11 is actually weaker than Theorem 2.3 since it requires stronger conditions on σ. The reader may therefore wonder why we have expended so much effort on it. However, in our opinion, the approach of Xu *et al.* offers at least two attractive features that make it worthy of serious study. First, it is a *direct* mutlivariate approach, avoiding the 'tensor product problem' discussed in Section 2.1. It thus gives some insight as to how the linear functionals corresponding to the weights in the first layer of the network might distribute information to the hidden nodes. Second the approach offers at least hope of providing sharper estimates for smooth f. This is because it is based on well understood principles of convolution and quadrature.

2.3. Radial basis networks

In Section 2.1 we met some interesting connections between neural networks and radial functions. Radial functions can also be employed more directly in neural computation. Look again at the function (2.2) which represents the function computed by a multilayer perceptron with one hidden layer. As we have already considered, the c_j quantities can be considered roughly as thresholds; they raise or lower the value at which the sigmoidal function σ switches from its asymptotic 0 value at negative arguments to its 1 value

at positive arguments. The more important part of the argument is the inner product $\mathbf{w}_j^T\mathbf{x}$. Assuming that the input vectors $\mathbf{x}$ all have a similar normalization, we see that the input to the network node corresponding to the term $\sigma(\mathbf{w}_j^T\mathbf{x} + c_j)$ depends on the projection of the input vector $\mathbf{x}$ onto the weight vector $\mathbf{w}_j$. In other words the weight vectors represent 'test vectors' or in artificial intelligence language 'features' against which each input vector $\mathbf{x}$ is tested for a match. The closer the alignment, the higher the response. Perceptron type networks are essentially row projection networks, a fact which we will investigate in much more detail in Section 3. For the moment, however, let us consider a different approach.

Suppose that instead of identifying features as row vectors, we identify them as points. In a classification problem each point $\mathbf{w}_j$ might be selected as a typical representative of a known class, or if we do not know suitable *a priori* classes they might simply be distributed in some sensible fashion about the input space. Instead of measuring the projection of $\mathbf{x}$ onto each $\mathbf{w}_j$, we consider the distance. We still have a network with the same topology as Figure 5, but now the input to a given unit is $\|\mathbf{x} - \mathbf{w}_j\|$. The function computed by the network becomes

$$g(\mathbf{x}) = \sum_j^k a_j \sigma(\|\mathbf{x} - \mathbf{w}_j\|). \tag{2.22}$$

Note that the meaning of the second layer weight a_j remains unchanged. We therefore have a linear combination of *radial basis functions.* The corresponding network architecture is known as a *radial basis network.* There is an extensive literature of approximation by radial basis functions: see Powell (1992). Indeed this theory is much more advanced and better understood than that of ridge function approximation so it is superfluous to go into details here. We will just explain how radial basis networks are normally applied. The architecture was first introduced by Broomhead and Lowe (1988) and this article remains a good explanation of the basic method. However, certain *caveats* need to be made in referring to this article now. First, it was believed at the time of writing that multilayer perceptrons required *two* hidden layers to solve the classification problem. The radial basis architecture was proposed partly as a solution to this problem. However, as we have already seen, it is now known that only one hidden layer is needed, at least to obtain density. (The question of whether more layers give a better degree of approximation is still open, although many authors believe that the answer will turn out to be affirmative.) In the model of Broomhead and Lowe we need to choose the centres $\mathbf{w}_j$: they cannot be learned (at least not without resorting to a nonlinear algorithm which is precisely what the authors wished to avoid). Only the a_j are adapted in the fitting process. One could, in principle, choose the weight vectors in (2.2) and therefore solve a

linear problem for those a_j in exactly the same way. Furthermore some of the comments of the authors on the backpropagation method could be construed as slightly misleading: we will come back to this in the next section. But notwithstanding these cautions, the idea remains a good one in view of the well developed theory and good numerical properties of radial basis functions. The method has retained its adherents. Mason and Parks (1992) survey this topic in a little more detail, and consider some more recent work.

The actual application is straightforward. First we need to choose a suitable activation function σ. Broomhead and Lowe recommend either the Gaussian $\sigma(r) = \exp(-r^2)$ or a multiquadric $\sigma(r) = (c_2 + r^2)^{1/2}$. We then assume that our function f that we wish to approximate is given at a set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$: this is the usual situation in classification problems (see the introductory remarks at the start of Section 2). The a_j are chosen simply to minimize

$$\sum_{j=1}^{t} (f(\mathbf{x}_j) - g(\mathbf{x}_j))^2.$$

This is a standard least-squares problem, which Broomhead and Lowe suggest is solved by computing the Moore–Penrose pseudoinverse (see, e.g., Ben-Israel and Greville (1974)). For simple networks the problem can be solved explicitly: the authors discuss the XOR problem (see Section 1.1) in some detail.

2.4. The effect of rounding on the approximation

Our work so far assumes that the weights can be evaluated to arbitrary precision. In a practical network, especially if implemented in hardware, one may only be able to store them to eight or 16 bits. There may be no point in using a very accurate network if its realization introduces large errors.

This problem has been encountered by various authors: the most systematic treatment would appear to be that of Brause (1992). Finite precision machine arithmetic is a classical issue in numerical analysis, but little deep theoretical work has been done in this context. The approach of Brause and others is largely experimental, backed up with some heuristic considerations and simple analysis. Brause's paper is a little off putting to numerical analysts at first sight, since it is expressed in the language of information theory. However, it is actually not difficult to come to grips with. His idea is to measure the error for a network with a given fixed system information, by which he means that a fixed *total* number of bits may be used to express the weights. So if we attempt to improve the approximation by introducing more neurons (and hence more weights) we must pay for that by storing the weights to a lower precision. He then takes two test problems and computes

approximations (using an approximate minimax criterion) for fixed information, trading off the number of weights against the precision. He discovers that for each test problem there is a well defined optimum precision giving the best achievable error (a result which will not surprise anyone who has attempted to use finite differences to compute derivatives!).

3. Numerical analysis of learning algorithms

We now turn our attention to some algorithmic aspects of neural networks. We are going to consider the so-called *supervised* learning problem which we will pose as follows (see also Section 2.3).

We are given at a set of points or *patterns* $\{\mathbf{x}_1, \ldots \mathbf{x}_t\}$ in $\mathbb{R}^n$. Associated with each pattern $\mathbf{x}_j$ there is a desired response y_j. For simplicity we will assume here that the network is to have a single output so y_j is a scalar. (For the single layer perceptron to be considered first, we shall show that this involves no loss of generality: the outputs can be treated separately as described in Section 3.1. For multilayer networks this is not the case, but nevertheless the single output case is sufficient to illustrate the situation.) We think of the y_j as being values of some function f which we want the network to 'learn': it is supposed to identify some generic features of the mapping so that when presented with an unknown input $\mathbf{x}$ it will estimate the corresponding y. In mathematical terms we would think of this process as interpolation (or extrapolation), but in learning theory it is called *generalization.* To start with, we need some measure of error. In Section 2 we used $\|\cdot\|_\infty$, but this is unlikely to lead to easy algorithms. Moreover the fact that we are working in $\mathbb{R}^n$ and using inner products suggest using a Hilbert space formulation. Of course, this is not the only choice: entropic measures also have their advocates particularly among the information theory fraternity. (See, e.g., Bichsel and Seitz (1989).) However, the basic simplicity of the least-squares approach means that it remains the most popular. So we formulate our learning problem as one of least-squares optimization. Suppose the actual output of our network for a given set of parameters is $g(\mathbf{x})$. For example, for a one-hidden-layer perceptron we have (2.2) with the parameters a_j, $\mathbf{w}_j$ and c_j, $j = 1, \ldots, k$ to be chosen. Then our task is simply to minimize $\sum(y_j - g(\mathbf{x}_j))^2$. Here the sum is over the t patterns $\mathbf{x}_j$, $j = 1, \ldots, t$. (For the case of nonscalar output we need to sum over the outputs as well.) As is usual in such problems we do not necessarily need a true minimum: a 'good' solution will do us. (Before getting down to work on this problem it is worth mentioning the *unsupervised learning problem.* In this case we do not know the desired outputs but wish to cluster the patterns into subsets apparently sharing common features. The most successful approach to this problem seems to be that of Kohonen: see Wasserman (1989, Ch. 4).)

Now having defined our problem a few general words on learning algorithms are in order. First, as expressed in the previous paragraph, the unsupervised learning problem is the classical one of nonlinear least squares. If n is not too large, it can be (and often has been) treated by standard optimization techniques. On the other hand, for very difficult problems simulated annealing or genetic algorithms can be applied. But we propose to look at the classic learning algorithms of the delta rule/backpropagation family, which remain the most popular approaches at least at present. An essential feature of these is that we only permit the patterns $\mathbf{x}_j$ to be presented to the system sequentially: we do not have them there all at once. Apart from the pragmatic consideration of popularity, there are two reasons for this restriction, one philosophical and one practical.

- First, mammals do not generally learn by considering a whole set of data at once. They learn from examples presented sequentially. Humans are not very good at considering lots of cases at once, and as far as we can tell, animals cannot do it at all. Yet they *do* successfully learn. To some extent, our least-squares error criterion is justified as a model of learning by the fact that it can tolerate this sequential restriction.
- Second, for very large n, second-order methods may simply be beyond the capabilities of the available hardware. Serial hardware may lack sufficient memory or computing power, and massively parallel machines present severe implementation problems. Sequential learning algorithms are memory efficient and naturally parallel: as such they deserve wider consideration even for problems which are not naturally formulated as 'learning': row projection methods are perhaps due for a renaissance!

Now it may be that in the long term, stochastic algorithms will prove more biologically plausible. But all of these are basically gradient descent methods with 'tricks' to avoid local minima. (Sometimes, particularly with genetic algorithms, the tricks are very sophisticated but the generalization remains valid.) So analysis of the simpler deterministic algorithms will not be wasted. Basically such analysis amounts to study of the underlying search geometry of the least-squares problem, which applies equally to the stochastic versions.

Finally, a remark to sceptical numerical analysts! It is often stated that backpropagation is 'merely steepest descent' and therefore unworthy of consideration by serious mathematicians. We shall show here that while it is certainly a gradient descent method, it is *not* steepest descent. In fact it has much better stability properties than steepest descent; one reason indeed for considering its use more widely!

The work described here first appeared in Ellacott (1990, 1992, 1993b,c). We mostly consider versions of the linear delta rule algorithm. However

a justification of this linearization as an approximation to the nonlinear backpropagation method will be included.

3.1. The delta rule

We begin by considering the simplest of all neural models, the basic perceptron. Figure 2 shows a perceptron with a single output. We will briefly consider the case of a multiple output perceptron, so the output is a vector and the weights form a matrix. To avoid a superfluity of subscripts, denote the training vectors (generically) by $\mathbf{x}$ and *desired* output vectors by $\mathbf{y}$. We will ignore the threshold c and instead treat the problem as one of approximation. (It is possible to make c learnable as well by including an extra input fixed at 1, but we need not consider this here.) If we can approximate $\mathbf{y}$ sufficiently well by the network, then obviously a suitable choice of c will solve the classification problem if this is what we are interested in. Let W be the weight matrix. In summary, then, we wish to find W such that $\mathbf{y} \approx W\mathbf{x}$ for all pairs $(\mathbf{x},\mathbf{y})$ of patterns and corresponding outputs. In general it is impossible to satisfy this exactly, so we seek a W for which the result holds approximately. The idea of a *learning algorithm* is as follows: we supply a set $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t\}$ of input patterns in $\mathbb{R}^n$ and for each $\mathbf{x}_i$ we supply the corresponding output $\mathbf{y}_i$. The system uses these pattern pairs to update its estimate of the desired weight matrix W. As we have already remarked, a learning algorithm is at heart simply an optimization process, but it has the special feature that the patterns are supplied serially rather than simultaneously as in standard least-squares approximation and optimization. The process of applying a learning algorithm is called *training* the network. Once training is complete, the system will be presented with previously unknown patterns $\mathbf{x}$ and used to predict the corresponding $\mathbf{y}$.

Although not the original perceptron algorithm, the following method known as the *delta rule* is generally accepted as the best way to train such a network. We assume initially that W is updated after each training pattern. The change in W when the pattern $\mathbf{x}$ is presented is given (Rumelhart and McClelland, 1986, p. 322) by

$$(\delta W)_{ji} = \eta(y_j - (W\mathbf{x})_j)\mathbf{x}_i,$$

where η is a parameter to be chosen called the *learning rate*, and $(W\mathbf{x})_j$ denotes the jth element of $W\mathbf{x}$. Thus if the error term in brackets is (say) positive, we will add a component of $\mathbf{x}$ to each row of W, increasing the output of the network for this pattern. Conversely if the error is negative, a component of $\mathbf{x}$ is subtracted, reducing the output for this pattern. In fact, we can simplify matters here by observing that there is no coupling between the rows of W in this formula: the new jth row of W depends *only* on the old jth row. This enables us to drop the subscript j, denoting y_j just by y,

and the jth row of W by the vector $\mathbf{w}^T$. Hence without loss of generality we return to the single output perceptron (Figure 2). We get

$$\delta w_i = \eta(y - \mathbf{w}^T\mathbf{x})x_i,$$

so

$$\delta\mathbf{w} = \eta(y - \mathbf{w}^T\mathbf{x})\mathbf{x}.$$

Thus given a current iterate weight vector $\mathbf{w_k}$,

$$\begin{aligned} \mathbf{w_{k+1}} &= \mathbf{w_k} + \delta\mathbf{w_k} \\ &= \mathbf{w_k} + \eta(y - \mathbf{w_k}^T\mathbf{x})\mathbf{x} \\ &= (I - \eta\mathbf{x}\mathbf{x}^T)\mathbf{w_k} + \eta y\mathbf{x}. \end{aligned} \tag{3.1}$$

The final equation is obtained by transposing the (scalar) quantity in brackets. Note the bold subscript $\mathbf{k}$ here, denoting the kth iterate, not the kth element. Observe also that the second equation makes clear what the delta rule actually does: it adds a suitable multiple of the current pattern $\mathbf{x}$ to the current weight vector (compare the discussion in Section 2.3). It is possible to analyse this iteration in the asymptotic case as $\eta \to 0$, but it is not used this way in practice. It is more relevant to consider a fixed η (Ellacott, 1990). We now prove some results about this iteration: the first lemma is a special case of a well known result (see, e.g., Oja (1983, p. 18)). The proof is a direct verification.

Lemma 3.1 Let $B = (I - \eta\mathbf{x}\mathbf{x}^T)$. Then B has only two distinct eigenvalues: $1 - \eta\|\mathbf{x}\|^2$ corresponding to the eigenvector $\mathbf{x}$ and 1 corresponding to the subspace of vectors orthogonal to $\mathbf{x}$. (Here $\|\cdot\|$ denotes the usual Euclidean norm.)

As an immediate consequence (see Isaacson and Keller (1966, p. 10, equation (11))) we obtain

Lemma 3.2 Provided $0 \leq \eta \leq 2/\|\mathbf{x}\|^2$, we have $\|B\| = \rho(B) = 1$, where $\rho(B)$ is the spectral radius of B.

Now suppose we actually have t pattern vectors $\mathbf{x}_1, \ldots, \mathbf{x}_t$. We will assume temporarily that these span the space of input vectors, i.e. that the set of pattern vectors contains n linearly independent ones. (This restriction will be removed later.)

Now for each pattern vector $\mathbf{x_p}$, we will have a different matrix B, say $B_p = (I - \eta\mathbf{x_p}\mathbf{x_p}^T)$. Let $\Lambda = B_t B_{t-1} \ldots B_1$.

Lemma 3.3 If $0 < \eta < 2/\|\mathbf{x_p}\|^2$ holds for each training pattern $\mathbf{x_p}$, and if the $\mathbf{x_p}$ span, then $\|\Lambda\| < 1$.

Proof. By definition, there exists $\mathbf{v}$ such that $\|\Lambda\| = \|\Lambda\mathbf{v}\|$ and $\|\mathbf{v}\| = 1$.

Thus $\|\Lambda\| = \|B_t B_{t-1} \dots B_1 \mathbf{v}\| \leq \|B_t B_{t-1} \dots B_2\| \|B_1 \mathbf{v}\|$ (from the definition of the norm). We identify two cases:

1 If $\mathbf{v}^T \mathbf{x}_1 \neq 0$, $\|B_1 \mathbf{v}\| < 1$, since the component of $\mathbf{v}$ in the direction of $\mathbf{x}$ is reduced (see Lemma 3.1: if this is not clear write $\mathbf{v}$ in terms of $\mathbf{x}$ and the perpendicular component, and apply B_1 to it.) On the other hand $\|B_t B_{t-1} \dots B_2\| \leq \|B_t\| \|B_{t-1}\| \dots \|B_2\| = 1$.

2 If $\mathbf{v}^T \mathbf{x}_1 = 0$, then $B_1 \mathbf{v} = \mathbf{v}$ (Lemma 3.1). Hence

$$\|\Lambda\| = \|B_t B_{t-1} \dots B_2 \mathbf{v}\|$$

and we may carry on removing Bs until Case 1 applies. Note that $\mathbf{v}$ cannot be orthogonal to all the $\mathbf{x_p}$ since by hypothesis they span. □

A common way to apply the delta rule is to apply patterns $\mathbf{x_1}, \mathbf{x_2}, \dots \mathbf{x_t}$ in order, and then to start again cyclically with $\mathbf{x}_1$. The presentation of one complete set of patterns is called an *epoch*. Assuming this is the strategy employed, iteration (3.1) yields

$$\mathbf{w_{k+t}} = \Lambda \mathbf{w_k} + \eta \mathbf{h} \tag{3.2a}$$

where Λ is as defined above and

$$\mathbf{h} = y_1 (B_t B_{t-1} \dots B_2) \mathbf{x_1} + \dots y_{t-1} B_t \mathbf{x_{t-1}} + y_t \mathbf{x_t}. \tag{3.2b}$$

Here, of course, y_p denotes the target y value for the pth pattern, not the pth element of a vector. Note that the Bs and hence $\mathbf{h}$ depend on η and the $\mathbf{x}$s, but *not* on the current $\mathbf{w}$.

Since δW in the delta rule is proportional to the error in the outputs, we get a fixed point of (3.1) only if all these errors can be made zero, which obviously is not true in general. Hence the iteration (3.1) does not in fact converge in the usual sense. On the other hand, we have shown (Lemma 3.2) that provided the $\mathbf{x_p}$ span the space of input vectors, then for sufficiently small η, $\|\Lambda\| < 1$. Hence the mapping $\mathbf{F}(\mathbf{w}) = \Lambda \mathbf{w} + \eta \mathbf{h}$ satisfies

$$\|\mathbf{F}(\mathbf{w}) - \mathbf{F}(\mathbf{v})\| = \|\Lambda(\mathbf{w} - \mathbf{v})\| \leq \|\Lambda\| \|\mathbf{w} - \mathbf{v}\|,$$

i.e. it is contractive with contraction parameter $\|\Lambda\|$. It follows from the Contraction Mapping Theorem that the iteration (3.2a) does have a unique fixed point. Now if there exists a $\mathbf{w}$ that makes all the errors zero, then it is easy to verify that this $\mathbf{w}$ is a fixed point of (3.1) and hence also of (3.2a). Otherwise, (3.1) has no fixed points, and the fixed point of (3.2a) depends on η: we denote it by $\mathbf{w}(\eta)$. In the limit, as the iteration (3.1) runs through the patterns, it will generate a limit cycle of vectors $\mathbf{w_k}$ returning to $\mathbf{w}(\eta)$ after the cycle of t patterns has been completed.

Since $\mathbf{w}(\eta)$ is a fixed point of (3.2a) we have (writing $\mathbf{h} = \mathbf{h}(\eta)$ and $\Lambda = \Lambda(\eta)$ to emphasize the dependence)

$$\mathbf{w}(\eta) = \Lambda(\eta) \mathbf{w}(\eta) + \eta \mathbf{h}(\eta). \tag{3.3}$$

Now what can we conclude about $\mathbf{w}(\eta)$? Let us denote by $\mathbf{w}^*$ the weight vector $\mathbf{w}$ (unique since the $\mathbf{x_p}$ span) that minimizes

$$\epsilon^2 = \sum_{p=1}^{t} (y_p - \mathbf{w}^T \mathbf{x_p})^2. \tag{3.4}$$

Denote by X the matrix whose columns are $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, and let

$$L = XX^T = \sum_{p=1}^{t} \mathbf{x_p} \mathbf{x_p}^T. \tag{3.5}$$

Then $\mathbf{w}^*$ satisfies the normal equations

$$L\mathbf{w}^* = \sum_{p=1}^{t} y_p \mathbf{x_p} = \mathbf{h}(0).$$

The second equality follows from (3.2*b*), observing that all the B matrices tend to the identity as $\eta \to 0$. On the other hand from (3.3) we get

$$H(\eta)\mathbf{w}(\eta) = \mathbf{h}(\eta) \quad \text{where} \quad H(\eta) = (I - \Lambda)/\eta.$$

Since by hypothesis the patterns span, L^{-1} exists. We define the *condition number* $\kappa(L) := \|L^{-1}\|\|L\|$. Moreover L is symmetric and positive definite, so $\kappa(L)$ is equal to the ratio of the largest and smallest eigenvalues of L (compare Isaacson and Keller (1966, p. 10, equation (11))).

A standard result on the solutions of linear equations (Isaacson and Keller, 1966, p. 37) gives, provided $\|L - H(\eta)\| < 1/\|L^{-1}\|$,

$$\frac{\|\mathbf{w}(\eta) - \mathbf{w}^*\|}{\|\mathbf{w}^*\|} \leq \frac{\kappa(L)}{1 - \|L^{-1}\|\|L - H(\eta)\|_2} \left(\frac{\|\mathbf{h}(\eta) - \mathbf{h}(0)\|}{\|\mathbf{h}(0)\|} + \frac{\|L - H(\eta)\|}{\|L\|} \right)$$

but

$$\Lambda(\eta) = \prod_{p=1}^{t} (I - \eta \mathbf{x_p} \mathbf{x_p}^T)$$

and considering powers of η in this product we obtain

$$\begin{aligned} \Lambda(\eta) &= I - \eta \sum_{p=1}^{t} \mathbf{x_p} \mathbf{x_p}^T + \mathcal{O}(\eta^2) \\ &= I - \eta L + \mathcal{O}(\eta^2). \end{aligned}$$

Thus $H(\eta) = L + \mathcal{O}(\eta)$. Also an examination of the products in (3.2*b*) reveals $\mathbf{h}(\eta) = \mathbf{h}(0) + \mathcal{O}(\eta)$. Putting all this together gives most of the following theorem.

Theorem 3.4 Suppose that the pattern vectors $\mathbf{x_p}$ span $\mathbb{R}^n$, and that $\mathbf{w}^*$ is (as above) the weight vector which minimizes the least-squares error of the

outputs over all patterns. If the delta rule is applied with fixed η satisfying the condition of Lemma 3.3, then the weights will converge to a limit cycle. Let $\mathbf{w}(\eta)$ be any member of the limit cycle, then as $\eta \to 0$,

(a) $\|\mathbf{w}(\eta) - \mathbf{w}^*\| = \mathcal{O}(\eta)$.
(b) If $\epsilon(\eta)$ is the root-mean-square error corresponding to $\mathbf{w}(\eta)$ (see (3.4)), and ϵ^* is the corresponding error for $\mathbf{w}^*$, then $\epsilon(\eta) - \epsilon^* = \mathcal{O}(\eta^2)$.

Proof. Convergence to a limit cycle has already been established. (a) follows from the remarks immediately preceding the theorem. (b) is simply the observation that for a least-squares approximation problem, the vector of errors for the best vector $\mathbf{w}^*$ is orthogonal to the space of possible $\mathbf{w}$s, so an $\mathcal{O}(\eta)$ error in $\mathbf{w}^*$ yields only an $\mathcal{O}(\eta^2)$ increase in the root mean square error. □

Unfortunately the rate of convergence is proportional to $\kappa(L)$, and as we shall see in the next subsection this can be large.

Finally we need to consider what happens when the $\mathbf{x_p}$ do not span the input pattern space. In this case it follows from Lemma 3.1 that the iteration (3.1) leaves the orthogonal complement of the span invariant. By decomposing the input space into the span and its orthogonal complement, a straightforward modification of the argument above shows that (3.2) is contractive on the span of the input patterns, so we still get convergence to a limit cycle. However, then L fails to be invertible, and discussion of the behaviour as $\eta \to 0$ requires examination of the singular vectors of L.

An interesting sidelight on the argument above is to consider what would happen if we presented the patterns during each epoch in *random* order, while still insisting that the whole set of patterns is presented precisely once during the epoch. With this approach, each epoch would still give a contractive mapping, but this mapping would be different for each choice of order. Since there is a large but finite number of such permutations of order, we have an iterated function scheme (Falconer, 1990, Ch. 9). We may expect a fractal attractor, even though this iteration is linear.

3.2. The 'Epoch Method'

Since we are assuming that we have a fixed and finite set of patterns $\mathbf{x_p}, \mathbf{p} = 1, \ldots t$, an alternative strategy is not to update the weight vector until the whole epoch of patterns has been presented. This idea is initially attractive since we shall see that this actually generates the steepest-descent direction for the least-squares error. We will call this the *epoch method* to distinguish it from the usual delta rule. This leads to the iteration

$$\mathbf{w_{k+1}} = \mathbf{w_k} - \eta \sum_{p=1}^{t} (\mathbf{x_p x_p^T}) \mathbf{w_k} + \eta \sum_{p=1}^{t} (y_p \mathbf{x_p})$$

$$= \Omega \mathbf{w_k} - \eta \sum_{p=1}^{t} (y_p \mathbf{x_p}), \tag{3.6}$$

where

$$\Omega = (I - \eta X X^T) = (I - \eta L).$$

(3.6) is, of course, the equivalent of (3.2a), not (3.1), since it corresponds to a complete epoch of patterns. There is no question of limit cycling, and, indeed, a fixed point will be a true least-squares minimum $\mathbf{w}^*$. To see this, put $\mathbf{w_{k+1}} = \mathbf{w_k} = \mathbf{w}^*$ and observe that (3.6) reduces to the normal equations for the least-squares problem. Moreover, the iteration (3.6) is simply steepest descent for the least-squares problem, applied with a fixed step length. Unfortunately, however, there is a catch! To see what this is, we need to examine the eigenvalues of Ω.

Clearly $L = XX^T$ is symmetric and positive semidefinite. Thus it has real nonnegative eigenvalues. In fact, provided the $\mathbf{x_p}$ span, it is (as is well known) strictly positive definite. The eigenvalues of Ω are $(1 - \eta) \times$ (the corresponding eigenvalues of L), and for a strictly positive definite matrix all the eigenvalues must be strictly positive. Thus we have for η sufficiently small, $\rho(\Omega) = \|\Omega\| < 1$.

Hence the iteration (3.6) will converge, provided the patterns span and η is sufficiently small. But how small does η have to be? (Recall that for the usual delta rule we need only the condition of Lemma 3.3.) To answer this question we need more precise estimates for the spectrum of L and the norm of Ω. From these we will be able to see why the epoch algorithm does not always work well in practice.

Suppose $L = XX^T$ has eigenvalues λ_j, $j = 1 \ldots n$, with

$$0 < \lambda_n \leq \lambda_{n-1} \leq \ldots \leq \lambda_1 = \rho(XX^T) = \|XX^T\| = \|X^T\|^2.$$

The eigenvalues of Ω are $(1 - \eta\lambda_1) \leq (1 - \eta\lambda_2) \leq \ldots \leq (1 - \eta\lambda_n)$, and $\rho(\Omega) = \max\{|1 - \eta\lambda 1|, |1 - \eta\lambda_n|\}$. (Observe that Ω is positive definite for small η, but ceases to be so when η becomes large.) Now

$$\lambda_1 = \|X^T\|^2 = \max_{\|\mathbf{v}\|=1} \|X^T\mathbf{v}\|^2 = \max_{\|\mathbf{v}\|=1} \mathbf{v}^T X X^T \mathbf{v} \leq \sum_{p=1}^{t} \|\mathbf{x_p}\|^2. \tag{3.7}$$

On the other hand, we can get a lower bound by substituting a particular $\mathbf{v}$ into the expression on the right-hand side of (3.7). For instance, we have for any k, $k = 1, \ldots, t$,

$$\lambda_1 \geq \frac{1}{\|\mathbf{x_k}\|} \left(\sum_{p=1}^{t} \mathbf{x_k}^T \mathbf{x_p} \right)^2 \geq \|\mathbf{x_k}\|. \tag{3.8}$$

Now consider a particular case.

Suppose the $\mathbf{x_p}$ cluster around two vectors $\mathbf{u}$ and $\mathbf{v}$ which are mutually ortho*normal*. If these represent two classes which are to be separated, we are in an ideal situation for machine learning: the pattern classes are in two widely separated convex sets. However, even in this case the behaviour of the epoch method is not good. If the clusters are of equal size, we have from the first inequality in (3.8)

$$\lim_{\epsilon \to 0} \lambda_1 \geq /2 \quad \text{and since the rank of } L = XX^T \text{ collapses to 2,} \quad \lim_{\epsilon \to 0} \lambda_n = 0.$$

Thus, unlike the ordinary delta rule for which the convergence condition depends only on the norm of the individual patterns, for the epoch method (i.e. steepest descent) we may require an arbitrary small μ to get convergence. As promised we have shown that the delta rule is much more stable.

3.3. Generalization to nonlinear systems

As we saw in Section 1, the usefulness of linear neural systems is limited, since many pattern recognition problems are not linearly separable. We need to generalize to nonlinear systems such as the backpropagation algorithm for the multilayer perceptron. Clearly we can only expect this type of analysis to provide a local result: global behaviour is likely to be more amenable to dynamical systems or control theory approaches. Nevertheless, a local analysis can be useful in discussing the asymptotic behaviour near a local minimum.

The obvious approach to this generalization is to attempt the 'next simplest' case, i.e. the backpropagation algorithm. However, this method looks complicated when written down explicitly: in fact much more complicated than it actually is! A more abstract line of attack turns out to be both simpler and more general. We will define a general nonlinear delta rule, of which backpropagation is a special case. For the linear network the dimension of the input space and the number of weights are the same: n in our previous notation. Now we will let M denote the *total number* of weights and n the input dimension.

So the input patterns $\mathbf{x}$ to our network are in $\mathbb{R}^n$, and we have a vector $\mathbf{w}$ of parameters in $\mathbb{R}^M$ describing the particular instance of our network: i.e. the vector of synaptic weights. For a single layer perceptron with m outputs, the 'vector' $\mathbf{w}$ is the the $m \times n$ weight matrix, and thus $M = mn$. For a multilayer perceptron, $\mathbf{w}$ is the Cartesian product of the weight matrices in each layer. For a general system with m outputs, the network computes a function $g : \mathbb{R}^M \times \mathbb{R}^n \to \mathbb{R}^m$. Say

$$\mathbf{v} = \mathbf{g}(\mathbf{w}, \mathbf{x}),$$

where $\mathbf{v} \in \mathbb{R}^m$. We equip $\mathbb{R}^M$, $\mathbb{R}^m$ and $\mathbb{R}^n$ with the Euclidean norm. For

pattern $\mathbf{x_p}$, denote the corresponding output by $\mathbf{v_p}$, i.e.

$$\mathbf{v_p} = \mathbf{g}(\mathbf{w}, \mathbf{x_p}).$$

We assume that $\mathbf{g}$ is Frechêt differentiable with respect to $\mathbf{w}$, and denote by $D = D(\mathbf{w}, \mathbf{x})$ the $m \times M$ matrix representation of the derivative with respect to the standard basis. Readers unfamiliar with Frechêt derivatives may prefer to think of this as the gradient vector: for $m = 1$ it is precisely the row vector representing the gradient when $\mathbf{g}$ is differentiated with respect to the elements of $\mathbf{w}$. Thus, for a small change $\delta\mathbf{w}$ and fixed $\mathbf{x}$, we have (by the definition of the derivative)

$$\mathbf{g}(\mathbf{w} + \delta\mathbf{w}, \mathbf{x}) = \mathbf{g}(\mathbf{w}, \mathbf{x}) + D(\mathbf{w}, \mathbf{x})\delta\mathbf{w} + o(\|\delta\mathbf{w}\|). \tag{3.9}$$

On the other hand for given $\mathbf{w}$, corresponding to a particular pattern $\mathbf{x_p}$, we have a desired output $\mathbf{y_p}$ and thus an error ϵ_p given by, say,

$$\epsilon_p^2 = (\mathbf{y_p} - \mathbf{v_p})^T(\mathbf{y_p} - \mathbf{v_p}) = \mathbf{q_p^T q_p}. \tag{3.10}$$

The total error is obtained by summing the ϵ_p^2s over the t available patterns, thus

$$\epsilon^2 = \sum_{p=1}^{t} \epsilon_p^2.$$

An ordinary descent algorithm will seek to minimize ϵ^2. However, the class or methods we are considering generate, not a descent direction for ϵ^2, but rather successive steepest descent directions for ϵ_p^2. Now for a change $\delta\mathbf{q_p}$ in $\mathbf{q_p}$ we have from (3.10)

$$\begin{aligned} \delta\epsilon_p^2 &= (\mathbf{q_p} + \delta\mathbf{q_p})^T(\mathbf{q_p} + \delta\mathbf{q_p}) - \mathbf{q_p^T q_p} \\ &= 2\delta\mathbf{q_p^T q_p} + \delta\mathbf{q_p^T}\delta\mathbf{q_p}. \end{aligned}$$

Since $\mathbf{y_p}$ is fixed,

$$\delta\mathbf{q_p} = -\delta\mathbf{v_p} = -D(\mathbf{w}, \mathbf{x_p})\delta\mathbf{w} + o(\|\delta\mathbf{w}\|) \quad \text{by (3.9)}.$$

Thus

$$\begin{aligned} \delta\epsilon_p^2 &= -2(D(\mathbf{w}, \mathbf{x_p})\delta\mathbf{w})^T(\mathbf{y_p} - \mathbf{g}(\mathbf{w}, \mathbf{x_p})) + o(\|\delta\mathbf{w}\|) \\ &= -2\delta\mathbf{w}^T(D(\mathbf{w}, \mathbf{x_p}))^T(\mathbf{y_p} - \mathbf{g}(\mathbf{w}, \mathbf{x_p})) + o(\|\delta\mathbf{w}\|). \end{aligned}$$

Hence, ignoring the $o(\|\delta\mathbf{w}\|)$ term, and for a fixed size of small change $\delta\mathbf{w}$, the largest decrease in ϵ_p^2 is obtained by setting

$$\delta\mathbf{w} = \eta(D(\mathbf{w}, \mathbf{x_p}))^T(\mathbf{y_p} - \mathbf{g}(\mathbf{w}, \mathbf{x_p})).$$

This is the generalized delta rule. Compare this with the single output linear perceptron, for which the second term in this expression is scalar with

$$\mathbf{g}(\mathbf{w}, \mathbf{x_p}) = \mathbf{w}^T\mathbf{x}_p,$$

and the derivative is the gradient vector (considered as a row vector) obtained by differentiating this with respect to $\mathbf{w}$, i.e. $\mathbf{x_p}^T$. Thus we indeed have a generalization of (3.1). Given a kth weight vector $\mathbf{w_k}$, we have

$$\begin{aligned}\mathbf{w_{k+1}} &= \mathbf{w_k} + \delta\mathbf{w_k} \\ &= \mathbf{w_k} + \eta(D(\mathbf{w_k}, \mathbf{x_p}))^T(\mathbf{y_p} - \mathbf{g}(\mathbf{w_k}, \mathbf{x_p})). \end{aligned} \tag{3.11}$$

To proceed further, we need to make evident the connection between (3.11) and the analysis of Section 3.1. However, there is a problem in that, guided by the linear case considered above, we actually expect a limit cycle rather than convergence to a minimum. Nevertheless it is necessary to fix attention to some neighbourhood of a local minimum, say $\mathbf{w}^*$, of the least-squares error ϵ: clearly we cannot expect any global contractivity result as in general ϵ may have many local minima, as is well known in the backpropagation case. Now from (3.10) and (3.11) we obtain (assuming continuity and uniform boundedness of D in a neighbourhood of $\mathbf{w}^*$),

$$\begin{aligned}\mathbf{w_{k+1}} &= \mathbf{w_k} + \eta(D(\mathbf{w_k}, \mathbf{x_p}))^T(\mathbf{y_p} - \mathbf{g}(\mathbf{w}^*, \mathbf{x_p}) - D(\mathbf{w}^*, \mathbf{x_p})(\mathbf{w_k} - \mathbf{w}^*)) \\ &\quad + o(\|\mathbf{w_k} - \mathbf{w}^*\|) \\ &= (I - \eta D(\mathbf{w_k}, \mathbf{x_p})^T D(\mathbf{w}^*, \mathbf{x_p}))\mathbf{w_k} + \eta(D(\mathbf{w_k}, \mathbf{x_p}))^T \\ &\quad \times(\mathbf{y_p} - \mathbf{g}(\mathbf{w}^*, \mathbf{x_p}) + D(\mathbf{w}^*, \mathbf{x_p})\mathbf{w}^*) + o(\|\mathbf{w_k} - \mathbf{w}^*\|). \end{aligned} \tag{3.12}$$

The connection between (3.11) and (3.1) is now clear. Observe that the iteration matrix $(I - \eta D(\mathbf{w_k}, \mathbf{x_p})^T D(\mathbf{w}^*, \mathbf{x_p}))$ is not exactly symmetric in this case, although it will be nearly so if $\mathbf{w_k}$ is close to $\mathbf{w}^*$. More precisely, let us assume that $D(\mathbf{w}, \mathbf{x})$ is Lipschitz continuous at $\mathbf{w}^*$, uniformly over the space of pattern vectors $\mathbf{x}$. Then we have

$$\begin{aligned}\mathbf{w_{k+1}} &= (I - \eta D(\mathbf{w}^*, \mathbf{x_p})^T D(\mathbf{w}^*, \mathbf{x_p}))\mathbf{w_k} + \eta(D(\mathbf{w}^*, \mathbf{x_p}))^T \\ &\quad \times(\mathbf{y_p} - \mathbf{g}(\mathbf{w}^*, \mathbf{x_p}) + D(\mathbf{w}^*, \mathbf{x_p})\mathbf{w}^*) + \mathcal{O}(\|\mathbf{w_k} - \mathbf{w}^*\|). \end{aligned} \tag{3.13}$$

Suppose we apply the patterns $\mathbf{x_1}, \ldots \mathbf{x_t}$ cyclically, as for the linear case. If we can prove that the linearized part (i.e. what we would get if we applied (3.13) without $\mathcal{O}$ term) of the mapping $\mathbf{w_k} \to \mathbf{w_{k+t}}$ is contractive, it will follow by continuity that there is a neighbourhood of $\mathbf{w}^*$ within which the whole mapping is contractive. This is because, by hypothesis, we have only a finite number of patterns. To establish contractivity of the linear part, we may proceed as follows.

First observe that

$$D(\mathbf{w}^*, \mathbf{x_p})^T D(\mathbf{w}^*, \mathbf{x_p})$$

is positive semidefinite. Thus for η sufficiently small,

$$\|I - \eta D(\mathbf{w}^*, \mathbf{x_p})^T D(\mathbf{w}^*, \mathbf{x_p})\| \leq 1.$$

We may decompose the space of weight vectors into the span of the eigenvectors corresponding to zero and nonzero eigenvalues respectively. These spaces are orthogonal complements of each other, as the matrix is symmetric. On the former space, the iteration matrix does nothing. On the latter space it is contractive provided

$$\eta < 1/\rho(D(\mathbf{w}^*, \mathbf{x_p})^T D(\mathbf{w}^*, \mathbf{x_p})).$$

We may then proceed in a similar manner to Lemma 3.3, provided the contractive subspaces for each pattern between them span the whole weight space. (If this condition fails then a difficulty arises, since the linearized product mapping will have norm 1, so the nonlinear map could actually be expansive on some subspace. We will not pursue this detail here.) For the single output case, $D(\mathbf{w}^*, \mathbf{x_p})$ is just a row vector, and we can identify the eigenvectors explicitly as in Lemma 3.1.

The *backpropagation rule* (Rumelhart and McClelland, 1986, pp. 322–328) used in many neural net applications is a special case of this. The name backpropagation derives from the fact that for an multilayer perceptron, the necessary terms in this expression can be calculated recursively back from the top layer. However, this is not relevant to our analysis here. We should make it clear, however, that backpropagation is rarely used in this 'pure' form. Rumelhart and McClelland themselves advocate the use of a 'momentum term' which is somewhat analogous to the Levenberg Marquardt method used in classical optimization (Moré, 1978). Moreover, the literature abounds with acceleration techniques. Fombellida and Destiné (1992) discuss two of the most popular: the *delta-bar-delta* and *quickprop* methods. They do some numerical comparisons and actually suggest a hybrid of the two methods as the most effective. However, none of the methods appears to have been subjected to any serious numerical analysis! A novel approach to accelerating backpropagation has been suggested by Almeida and Silva (1992). This is somewhat related to the work to be discussed in Section 3.4, so we defer consideration of this paper until Section 4.

3.4. The singular value decomposition and principal components

Since we now know that the backpropagation rule can be realistically considered as behaving locally like the delta rule, it makes sense to return to a closer study of the linear algorithm. Several interesting results can be obtained from Singular Value Decomposition (SVD). This is unsurprising in view of the well known connections between neural nets and statistical decision theory. Unfortunately they are easily obtained only for the algorithm in its 'epoch' form (3.6). This is a pity in view of the previous analysis, but since the algorithms are at least asymptotically the same for small η, they seem nevertheless worth having. Not all of the results in this section

are really new, but it is difficult to find a formal and coherent exposition of them in the literature. This attempt at a systematic description is thus worthwhile.

Firstly, we can provide a simple explanation for the well known phenomenon of *overgeneralization* reported in many practical studies with neural networks. This is the observation that better results may well be obtained if the iteration is *not* continued to convergence. These problems are closely related to the issue of nonspanning patterns which we have already encountered. In many network applications such as vision, we may have a very large number of free weights. For example, even a medium resolution 64×64 image will have 4096 pixels. If we feed this into the network without any compression we will have at least this many weights. If we are training the network to recognize (say) a certain object in a set of images, it is most unlikely that we will have enough data to prevent the problem being severely underdetermined. But in fact, the delta rule (even in epoch form) can cope with this if the number of iterations is restricted: it includes a kind of built in compression. Recall (3.6):

$$\mathbf{w_{k+1}} = \Omega \mathbf{w_k} - \eta \sum_{p=1}^{t} (y_p \mathbf{x_p}),$$

where $\Omega = (I - \eta X X^T)$. We decompose X in singular value form. (See, e.g., the chapters by Wilkinson and Dennis in Jacobs (1977, pp. 3–53 and 269–312) respectively. Also Chapter 6 of Ben-Israel and Greville (1974).) Specifically we may write

$$X = PSQ^T \tag{3.14}$$

where P and Q are orthogonal and S is diagonal (but not necessarily square). Recall that in this context $\mathbf{y}$ is not a single output vector but the vector of single outputs over all the patterns. We find

$$\begin{aligned} \mathbf{w_{k+1}} &= (I - \eta P S S^T P^T)\mathbf{w_k} - \eta X \mathbf{y}, \\ &= P(I - \eta S S^T) P^T \mathbf{w_k} - \eta P S Q^T \mathbf{y} \end{aligned}$$

or, with $\mathbf{z_k} = P^T \mathbf{w_k}$ and $\mathbf{u} = P^T \mathbf{y}$,

$$\mathbf{z_{k+1}} = (I - \eta S S^T)\mathbf{z_k} - \eta S \mathbf{u}. \tag{3.15}$$

At this point the notation becomes a little messy: let us denote by $(\mathbf{z_k})_i$ the ith element of $\mathbf{z_k}$. These elements are decoupled by SVD. More specifically, suppose X has r nonzero singular values (the diagonal elements of S) $\nu_1 \geq \nu_2 \geq \ldots \geq \nu_n$, (3.15) when written elementwise gives

$$(\mathbf{z_{k+1}})_i = (1 - \eta \nu_i^2)(\mathbf{z_k})_i - \eta \nu_i u_i, \quad \text{for } i = 1, \ldots, r$$

and

$$(\mathbf{z_{k+1}})_i = (\mathbf{z_k})_i \quad \text{for } i = r+1, \ldots, n.$$

Assuming that η is sufficiently small to guarantee convergence (i.e. all terms $(1 - \eta\nu_i^2) < 1$), it is easy to see that convergence will be very much faster for the $(\mathbf{z_k})_i$ corresponding to the larger singular values. This is exactly what we would like. Since P and Q are orthogonal matrices their rows and columns have norm 1. Thus we see from (3.15) that the large singular values correspond to the actual information in the pattern data X. (This approach is called *principal component analysis.*) The delta rule (in epoch form at least) has the nice property of converging on the principal components of the data *first.* Unfortunately it is very hard to tell from the iteration when this has occurred since small singular values can make a large contribution to the least-squares error. This explains the phenomenon of *overgeneralization.* Initially the iteration picks out significant features in the variability of the data. Continued iteration makes it try to separate insignificant features or noise.

In view of the problems of slow convergence and underdetermination, many authors have commented on the advisability of performing some preprocessing of the input patterns before feeding them to the network. Often (not always, of course) the preprocessing suggested is linear. At first sight this seems to be a pointless exercise, for if the raw input data vector is $\mathbf{x}$ with dimension n', say, the preprocessing operation is represented by the $n \times n'$ matrix T, W is the input matrix of the net and we denote by the vector $\mathbf{h}$ the input to the next layer of the net, then

$$\mathbf{h} = WT\mathbf{x}. \tag{3.16}$$

Obviously, *the theoretical representational power of the network is the same as one with unprocessed input and input matrix WT.* However, this does *not* mean that these preprocessing operations are useless. We can identify at least the following three uses of preprocessing.

1 to reduce work by reducing dimension and possibly using fast algorithms (e.g. the FFT or wavelet transform) (so we do not want to increase the contraction parameter in the delta rule iteration);
2 to improve the search geometry by removing principal components of the data and corresponding singular values that are irrelevant to the classification problem;
3 to improve the stability of the iteration by removing near zero singular values (which correspond to noise) and clustering the other singular values near to 1: i.e. in the language of numerical analysis to *precondition* the iteration.

We will not address all these three points explicitly here. Instead we will

derive some theoretical principles with the aid of which the issues may be attacked. The first point to consider is the effect of the filter on the stability of the learning process. For simplicity, we again consider only the linear epoch algorithm here.

We hope, of course, that a suitable choice of filter will make the learning properties better, but the results here show that whatever choice we make, the dynamics will not be made much worse unless the filter has very bad singular values. In particular, we show that if the filter is an orthogonal projection, then the gradient descent mapping with filtering will be at least as contractive as the unfiltered case.

We see from (3.6) that the crucial issue is the relationship between the unfiltered update matrix

$$\Omega = (I - \eta XX^T) \tag{3.17}$$

and its filtered equivalent

$$(I - \eta TXX^TT^T) = \Omega' \tag{3.18}$$

say.

Note that these operators may be defined on spaces of different dimension: indeed for a sensible filtering process we would expect the filter T to involve a significant dimension reduction. Recall that Ω in (3.17) is $n \times n$ and let us take Ω' to be $n' \times n'$. Note also that for purposes of comparison we have assumed the learning rates η are the same.

A natural question is to try to relate the norms of these two operators, and hence the rate of convergence of the corresponding iterations. As before, we suppose $L = XX^T$ has eigenvalues λ_j, $j = 1 \ldots n$, with

$$0 < \lambda_n \leq \lambda_{n-1} \leq \ldots \leq \lambda_1 = \rho(XX^T) = \|XX^T\| = \|X^T\|^2.$$

(Note here we assume the **x**s span so $\lambda_n \neq 0$. In terms of the singular values ν_i of X, $\nu_i^2 = \lambda_i$.)

We need to relate the eigenvalues of XX^T with those of $TXX^TT^T = L'$, say. Let L' have eigenvalues $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_{n'} > 0$ and T have singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n > 0$. Note that we are assuming T has full rank n' and so has no nonzero singular values. This is a reasonable assumption since there is no point in using a filter which has a nontrivial kernel. We should reduce the codomain dimension of the operator instead. For example, an orthogonal projection is formally defined as a mapping from (say) $\mathbb{R}^n$ to itself. However, in practice, if we use an orthogonal expansion as a filter, we will reduce the dimension by choosing an orthogonal basis for the image and ignoring the rest of the basis required to span $\mathbb{R}^n$.

Proposition 3.5 With the notation above, $\mu_1 \leq \sigma_1^2 \lambda_1$ and $\mu_{n'} \geq \sigma_{n'}^2 \lambda_n$.

Proof. The first inequality is straightforward. Since L and L' are symmetric

$$\mu_1 = \|TXX^TT^T\| \leq \|T\|\|XX^T\|\|T^T\| = \sigma_1^2\lambda_1.$$

The second inequality is slightly more difficult. Let $\mathbf{u_n}$ be the normalized eigenvector of L' corresponding to μ_n. Then

$$\mu_n = \mathbf{u_n}^T\mu_n\mathbf{u_n} = \mathbf{u_n}^TTXX^TT^T\mathbf{u_n} = \|X^TT^T\mathbf{u_n}\|^2.$$

But $\|X^TT^T\mathbf{u_n}\| \geq \lambda_n^{1/2}\sigma_{n'}$ as may be found by writing both matrices in terms of their SVDs. □

This result means that $\|\Omega'\|$ cannot be much larger than $\|\Omega\|$ if T has singular values close to 1.

Corollary 3.6 Let T be a truncated orthogonal expansion, or any other filter that is the restriction of an orthogonal projection to the orthogonal complement of its kernel (e.g. unweighted local averaging: see Ellacott (1993a)). Then with filtering applied the epoch method will converge at least as fast (as expressed by its contraction parameter) as the unfiltered version.

Proof. All the singular values of an orthogonal projection are either 0 (corresponding to the kernel) or 1 (corresponding to the image). It follows from Proposition 3.5 that the norm of Ω' in (3.18) cannot be greater than that of Ω in (3.17). □

The result above gives us some insight into the uses of filters for data compression, although its extension to the nonlinear case is not obvious: filters are applied to the input of a multilayer perceptron, whereas to employ this result directly we would need to apply them to the tangent space: compare (3.13). Let us turn now to the issue of preconditioning. An ideal choice of filter to act as a preconditioner would not require knowledge of the particular data set under consideration, but this would seem to be an almost impossible requirement since the matrix Ω is defined in terms of this data. The best one might hope for is something that would work for large classes of data sets in a particular context such as vision or speech recognition. In other words we might try to derive information from the problem domain, and use this to construct the filter. As an illustration of the difficulties, we show that the theoretically optimal preconditioner for the delta rule in epoch form is both easily described and completely useless! Suppose, as above, X has SVD PSQ^T. We set the filter matrix T to be the Moore–Penrose pseudoinverse of X (see, e.g., Ben-Israel and Greville (1974)) which we denote by $X^\#$. So

$$T = X^\# = QS^\#P^T.$$

Then

$$TX = QS^{\#}P^{T}PSQ^{T} = QS^{\#}SQ^{T}.$$

Thus (with the same notation as before Proposition 3.5)

$$L' = TXX^{T}T^{T} = QS^{\#}SS^{T}S^{\#T}Q^{T},$$

and $S^{\#}SS^{T}S^{\#T}$ is a diagonal matrix with diagonal elements either 0 or 1. Thus all the eigenvalues of L' are either 0 or 1 and, indeed, if the $\mathbf{x}$s span so that XX^{T} has no zero eigenvalues, then all the eigenvalues of L' are 1. With $\eta = 1$, the iteration will converge in a single iteration. This is not surprising, since once we know $X^{\#}$, the least-squares solution for $\mathbf{w}$ may be given explicitly! (For the nonlinear case we would need to compute the local pseudoinverses for the relevant tangent vectors.)

A modification of the approach which might be slightly more practicable is just to remove the large eigenvalues of XX^{T} based on computation of the dominant singular values, and corresponding singular vectors, of X. We present an algorithm for removing the principal components one at a time. Whether an approach based on removal of individual singular values is going to be very useful for the interesting case of very large n is debatable: it may help if the data matrix X is dominated by a few principal components with large singular values but otherwise it it likely to be too inefficient. However, the method does suggest ways forward. (A recent paper (Oja, 1992) also has some relevance to this problem, as does the method of Almeida and Silva (1992) which we consider in Section 4.1.)

The first stage is to compute the largest eigenvalue and corresponding eigenvector of XX^{T}. This may be carried out by the power method (Isaacson and Keller, 1966, p. 147) at the same time as the ordinary delta rule iteration: the computation can be performed by running through the patterns one at a time, just as for the delta rule itself. We get a normalized eigenvector $\mathbf{p_1}$ of XX^{T} corresponding to the largest eigenvalue λ_1 of XX^{T}. Set

$$T = I + (\lambda_1^{-1/2} - 1)\mathbf{p_1}\mathbf{p_1}^{T}.$$

A routine calculation shows that $TXX^{T}T^{T}$ has the same eigenvectors as XX^{T}, and the same eigenvalues but with λ_1 replaced by 1. Each pattern $\mathbf{x_p}$ should then be multiplied by T, and, since we are now iterating with different data, the current weight estimate $\mathbf{w}$ should be multiplied by T^{-1}. It is easy to check that

$$T^{-1} = I + (\lambda_1^{1/2} - 1)\mathbf{p_1}\mathbf{p_1}^{T}.$$

Basically the same idea can be used for the iteration with the weights updated after each pattern. However there is a problem in that the update matrix Λ is not exactly symmetric, although it is nearly so for small η. This could be overcome by computing the right as well as left eigenvectors of

$(\Lambda - I)/\eta$, but unfortunately this would require presenting the patterns in *reverse* order: somewhat inconvenient for a neural system. Another possibility is to perform two cycles of the patterns, with the patterns in reverse order on the second cycle. The composite iteration matrix $\Lambda^T\Lambda$ will then be symmetric. Although space and the requirements of simplicity do not permit a full discussion here, there is no reason in principle why this algorithm should not be applied to the nonlinear case.

4. Some numerical applications of neural networks

The most successful applications of neural networks have been in pattern recognition areas such as speech, vision and nonlinear control, where satisfactory existing models do not exist. The power of the approach is the ability of the network to construct its own model. However it is also possible to design networks to solve some standard mathematical problems. We will conclude our survey by looking at some of these. Of course, it is not suggested that these methods will out-perform standard algorithms when run on conventional machines. So why should we study these methods? First, as we saw in the previous section, there are close connections between linear algebra and the methods of filtering and data compression used in neural network applications. To design suitable filters we need networks to perform linear algebra calculations. Second, neural network algorithms are naturally parallel. They lend themselves easily to implementation on array processors. Moreover we do not in fact even need the power of current parallel machines. The whole point of neural networks is that they use large arrays of very simple nonprogrammable processors. Neural network chips are already starting to appear. When these become large enough and cheap enough it will become possible to design hard-wired circuitry to perform a range of standard tasks. Of particular interest are problems such as the travelling salesman problem which involve optimization on graphs. This problem is of course NP complete so we cannot guarantee to find an optimum solution. But it turns out that we can find good solutions quickly *if* we can build a large enough neural network. Thus in this last section we focus on the two issues of linear algebra and optimization.

4.1. Linear algebra applications

We first observe that the delta rule itself may be regarded as a row-projection method for solving linear equations in the least-squares sense. If we have a single output linear perceptron with pattern vectors $\mathbf{x_1}, \ldots, \mathbf{x_t}$ forming the columns of a matrix X, and corresponding required outputs $y_1, \ldots, y_t$, formed into a vector $\mathbf{y}$, then the delta rule will approximately minimize $\|\mathbf{y} - \mathbf{w}^T X\|$. Similarly if we use a t output net and make the output for $\mathbf{x_p}$ the pth column of I, the delta rule will approximately minimize $\|I - WX\|_S$,

The norm here is the Schur matrix norm: this is simply the square root of the sum of squares of all the elements. Thus the delta rule will attempt to invert X and will do so exactly if X is nonsingular, since then a true fixed point will exist (compare the remarks following (3.2)). Not only can we avoid storing the whole matrix X at once, but we can compute the rows of W individually as well.

Other networks and learning schemes have been proposed to perform various calculations concerned with least-squares approximation and principal components. There are several of these around, with variations on the basic ideas. We will look first at an example based on Baldi and Hornik (1989) but we modify it somewhat. Consider a multilayer perceptron with n inputs and n outputs. We have one hidden layer with m units. This network is illustrated in Figure 3. However for this network the activation of the hidden units (as well as the input and output units) is simply the identity function $\sigma(x) = x$. Thus if W is the matrix of weights in the first layer, and V is the matrix of weights in the second layer, the output for input vector $\mathbf{x}$ is simply $VW\mathbf{x}$. Now for any given $\mathbf{x}$ we specify our target output $\mathbf{y}$ as $\mathbf{y} = \mathbf{x}$. Obviously if we have $m \geq n$, then any V and W with $VW = I$ would achieve this exactly. But what happens for $m < n$? More specifically, let our input patterns $\mathbf{x}_j$, $j = 1, \ldots, t$, be the columns of an $n \times t$ matrix X. We train the network, perhaps by backpropagation, to minimize $\sum \|\mathbf{x_j} - \mathbf{g}(\mathbf{x_j})\|_2^2$, where the sum is over j and we recall that for any particular choice of V and W, $\mathbf{g}(\mathbf{x}) = VW\mathbf{x}$. Our minimization problem can be restated as follows: find V and W so that $\|X - VWX\|_S$ is minimized. As we have seen, training with a finite learning rate η will not in fact solve the least-squares problem exactly, but let us suppose that η is sufficiently small that for practical purposes we have the true minimum. (Since this is actually a linear problem there are no local minima in this case.) For convenience, we will also assume that the weight matrices are constrained so that $V = W^T$ (this restriction is not essential but it makes it easier to see what is going on). To understand what the matrix W does here, we write X in terms of its SVD $X = PSQ^T$ where P and Q are orthogonal and S is diagonal (but not necessarily square). The diagonal elements of S are the singular values $\nu_1 \geq \nu_2 \geq \ldots \geq \nu_n$. Suppose $\text{rank}(X) = r$ so that in fact $\nu_{r+1} \ldots \nu_n = 0$. The crucial stage is to find a matrix H satisfying $\text{rank}(H) \leq m$ and which minimizes $\|X - PHP^TX\|_S$. Once we have H it is not difficult to factorize PHP^T to get W and V. But for any H, (since the Schur norm of a matrix is unchanged if we multiply by an orthogonal matrix)

$$\begin{aligned} \|X - PHP^TX\|_S^2 &= \|(I - PHP^T)X\|_S^2 \\ &= \|(I - PHP^T)PSQ^T\|_S^2 \\ &= \|(P - PH)S\|_S^2 = \|(I - H)S\|_S^2 \end{aligned}$$

$$= \sum_{i=1}^{r} \nu_i^2 \Big\{ (1 - h_{ii})^2 + \sum_{j \neq i} h_{ji}^2 \Big\}.$$

Obviously, at the minimum H is diagonal. But we require $\text{rank}(H) \leq m$. Thus the minimizing H is obtained by setting $h_{ii} = 1$, $i = 1, \ldots, \min(r, m)$ and all the other elements to zero. If $r \leq m$, there is no loss of information in this process, and the patterns $\mathbf{x_p}$ are reconstructed exactly. If $r > m$, then the total error over all patterns is given by the square root of

$$\sum_{i=m+1}^{r} \nu_i^2. \tag{4.1}$$

It remains to perform the factorization $VW = PHP^T$. While the choice of H is unique, this is not so for the factorization. However, since PHP^T is symmetric, it makes sense to set $V = W^T$ as suggested above. In fact we have for the minimizing H,

$$HH^T = H$$

whence

$$PHP^T = PHH^TP^T = PH(PH)^T.$$

PH has (at most) m nonzero columns: we may take these as V and make $W = V^T =$ the first m rows of H^TP^T. Then $VW = PHP^T$ as required. The rows of W are those eigenvectors of XX^T corresponding to the largest singular values. The effect of W is to project the input patterns $\mathbf{x_p}$ onto the span of these vectors. If $r \leq m$ (which is certainly the case, for instance, if the number of patterns $t \leq m$) then the t n-vectors $\mathbf{x_p}$ are compressed by W onto m-vectors $\mathbf{y_p}$ with no loss of information, since we can recover X as W^TY. Here, of course, the columns of Y are the $\mathbf{y_p}$s. More usefully, even if $r > m$ there will be little loss in this compression provided the quantity (4.1) is small.

Moreover, by definition, VWX is a best rank m approximation to X in terms of the Schur norm, and can be constructed using the trained network by feeding the columns of X through the network one at a time to get the columns of VWX. It is well known and not hard to show that this matrix is also a best rank m approximation with respect to the matrix 2 norm: the proof is left for the reader!

A rather similar idea to that proposed in the last few paragraphs of Section 3 is used by Almeida and Silva (1992). However they propose using a separate perceptron to decorrelate the data. The weight matrix of this perceptron is the filter T so that the output is simply $T\mathbf{x}$. The aim is to find T such that $TXX^TT^T = I$ (compare (3.18)). Starting with $T = I$, they update T according to the rule

$$T_{n+1} = (1 + \alpha)T_n - \alpha(T_nLT_n^T)T_n \tag{4.2}$$

where, as in Section 3, $L = XX^T$. Now if L has full rank, certainly there exists a nonsingular T^* such that

$$T^* L T^{*T} = I. \tag{4.3}$$

To see this write $L = PDP^T$, where P is orthogonal, and let

$$T^* = D^{-1/2} P^T.$$

To see that it is a fixed point, simply set T^* satisfying (4.3) as T_n in (4.2). Almeida and Silva give as a sufficient condition for convergence

$$\alpha < \min\{\tfrac{1}{2}, (3\rho(L) - 1)^{-1}\}.$$

However, it is perhaps worth pointing out that convergence is sublinear, for let

$$F(T) = (1 + \alpha)T - \alpha(TLT^T)T \tag{4.4}$$

and suppose T^* satisfies (4.3). Direct calculation yields for $h \in \mathbb{R}$ and any $n \times n$ matrix S

$$\begin{aligned} F(T^* + hS) &= T^* + (1+\alpha)hS - \alpha T^* L T^{*T} hS - \alpha(hSLT^{*T}T^* \\ &\quad + hT^* L S^T T^*) + \mathcal{O}(h^2) \\ &= T^* + (1+\alpha)hS - \alpha hS - \alpha(hS + hT^* L^S T T^*) + \mathcal{O}(h^2) \end{aligned}$$

since T^* satisfies (4.3) and this condition also implies $LT^{*T}T^* = I$. Thus

$$\begin{aligned} F(T^* + hS) &= T^* + (1-\alpha)hS - \alpha h(T^{*T})^{-1} S^T T^* + \mathcal{O}(h^2) \\ &= (I + (1-\alpha)hS(T^*)^{-1} - \alpha h(T^{*T})^{-1} S^T) T^* + \mathcal{O}(h^2). \end{aligned}$$

Now if we choose S so that $(T^*)^{-1}S$ is antisymmetric we obtain

$$\begin{aligned} F(T^* + hS) &= (I + hS(T^*)^{-1})T^* + \mathcal{O}(h^2) \\ &= T^* + hS + \mathcal{O}(h^2). \end{aligned}$$

Thus we will not get linear convergence near the fixed point. This is not a true learning network anyway since all the patterns are required at once. Hence use of an optimization routine might be preferable. Or simply compute the SVD instead! Almeida and Silva do give a version which uses the patterns one at a time, but this requires a sequence of αs tending to zero. Thus it seems that a better update rule than (4.2) would be worth seeking. In spite of these problems the particular interest of this method is that Almeida and Silva have performed numerical experiments with networks interleaving ordinary backpropagation layers with layers trained by (4.2). Improved convergence of the backpropagation was found on two test problems.

We remark finally that another way to satisfy (4.3) is actually to compute the principal components of X. Oja *et al.* (1992) describe a network to do this.

4.2. Hopfield nets and graph optimization

The Hopfield net (see, e.g., Wasserman (1989, Ch. 6)) is a dynamic net which is of historical importance as the first usable nonlinear neural network. As such it was a significant factor in the revival of interest in connectionist models in the 1980s. The most important application of Hopfield nets today is in the field of graph theoretic optimization. We will first describe the Hopfield architecture, and then use it to address the travelling salesman problem.

Unlike the multilayer perceptron, the Hopfield net is not layered: all neuronal units are treated equally. Each is connected to every other unit with a *bidirectional* link. Thus, if the weight of the connection from unit i to unit j is w_{ij}, we have $w_{ij} = w_{ji}$. No connections are permitted from a unit to itself, i.e. $w_{ii} = 0$ (actually for neurons that can output only values 0 or 1, this is not a real restriction: for example see the discussion of the travelling salesman problem below). The topology of an n unit Hopfield net is thus the complete digraph on n vertices. In addition to taking input from the other nodes, each unit can also take a (constant) input. There are no output units as such: the output from each unit is formed into a state vector $\mathbf{x}$ (so the ith element of $\mathbf{x}$ is the output of the ith unit). Usually (and certainly for our application) $\mathbf{x}$ is a binary vector, i.e. each element is either 0 or 1. Since the connections of the network are recursive, it will evolve in time. In the basic Hopfield model, single neurons are 'fired' (i.e. updated) at a time, in random order (biologically plausible) or cyclically. (Sometimes later workers have chosen, on grounds of simplicity and speed, to update all the units at once in discrete time steps. As we shall see this does have disadvantages.) Let us form the weights into a (symmetric) matrix W and let $\mathbf{q}$ be the vector of inputs. At the kth time step we have a state vector $\mathbf{x_k}$. So at the $k+1$st time step, the input to the jth unit is the jth element of the vector $W\mathbf{x_k}+\mathbf{q}$. The activation of the unit is by thresholding: let p_j be the threshold on the jth unit and form these into a vector $\mathbf{p}$. Thus, denoting the jth element of a vector $\mathbf{y}$ by $(\mathbf{y})_j$, we fire the jth neuron by updating $(\mathbf{x_{k+1}})_j$ as

$$(\mathbf{x_{k+1}})_j = \begin{cases} 1 & \text{if } (W\mathbf{x_k} + \mathbf{q} - \mathbf{p})_j > 0, \\ 0 & \text{if } (W\mathbf{x_k} + \mathbf{q} - \mathbf{p})_j < 0, \\ (\mathbf{x_k})_j & \text{if } (W\mathbf{x_k} + \mathbf{q} - \mathbf{p})_j = 0. \end{cases} \tag{4.5}$$

(Sometimes other choices are made in the equality case, but this one is sensible: see below.) For single unit updating, we simply apply this rule for one particular j and leave the other elements unchanged. If we are updating all neurons at once, we apply it for all j.

Associated with the network state vector $\mathbf{x}$ is an energy functional

$$E(\mathbf{x}) = -\tfrac{1}{2}\mathbf{x}^T W\mathbf{x} + \mathbf{q}^T\mathbf{x} - \mathbf{p}^T\mathbf{x}, \tag{4.6}$$

which serves as a Liapunov function. Observe that although this is usually referred to as the energy, W is *not* positive definite. The condition $w_{ii} = 0$ guarantees that trace$(W) = 0$, so W must have negative eigenvalues. This applies equally to subspaces obtained by deleting rows and corresponding columns of W. Thus, the good news is that the minimum of E must occur not at a stationary point but at a vertex of the hypercube defined by the condition $0 \leq (\mathbf{x})_j \leq 1$, $j = 1, \ldots, n$, i.e. at a valid state vector $\mathbf{x}$. (So we can avoid the thresholding and use a continuous model instead, if we wish, but we will stick to the threshold model here.) This result is a special case of the Cohen–Grossberg Theorem, see e.g., Simpson (1990). The bad news is that we can get local minima.

A simple calculation shows that the change $\delta E = E(\mathbf{x_{k+1}}) - E(\mathbf{x_k})$ in the energy is given by

$$\delta E = -(W\mathbf{x_k} + \mathbf{q} - \mathbf{p})^T(\mathbf{x_{k+1}} - \mathbf{x_k}) - \tfrac{1}{2}(\mathbf{x_{k+1}} - \mathbf{x_k})^T W(\mathbf{x_{k+1}} - \mathbf{x_k}). \quad (4.7)$$

Now consider the inner product $(W\mathbf{x_k} + \mathbf{q} - \mathbf{p})^T(\mathbf{x_{k+1}} - \mathbf{x_k})$. Perhaps $(\mathbf{x_{k+1}} - \mathbf{x_k})_j = 0$ for some j, in which case no contribution to the inner product is made by the jth term. However, unless we have reached a stationary point of the iteration, there must be one or more js for which $(\mathbf{x_{k+1}} - \mathbf{x_k})_j \neq 0$. We identify two cases. Possibly, $(W\mathbf{x_k} + \mathbf{q} - \mathbf{p})_j > 0$. From (4.5) we must have $(\mathbf{x_{k+1}})_j = 1$ whence necessarily $(\mathbf{x}k)_j = 0$ and $(\mathbf{x_{k+1}} - \mathbf{x_k})_j = 1$. The jth term thus contributes a positive value to the inner product in this case. Conversely if $(W\mathbf{x_k} + \mathbf{q} - \mathbf{p})_j < 0$, $(\mathbf{x_{k+1}})_j = 0$, $(\mathbf{x_k})_j = 1$ and $(\mathbf{x_{k+1}} - \mathbf{x_k})_j = -1$. Again a positive contribution is made. Thus the first term of (4.7) constitutes a decrease in energy unless $(\mathbf{x_{k+1}} - \mathbf{x_k})_j = 0$. In this case, of course, we make no change in energy but moving on to another neuron will stop us getting stuck. Since W is not positive definite, we cannot guarantee that the second term in (4.4) is negative. One reason for firing single neurons, rather than updating them all at once, is that at most one element of $(\mathbf{x_{k+1}} - \mathbf{x_k})$, say the jth, can be nonzero. Then $(\mathbf{x_{k+1}} - \mathbf{x_k})^T W(\mathbf{x_{k+1}} - \mathbf{x_k}) = w_{jj} = 0$. Moreover by updating singly we will ultimately search in all possible directions from a given point until a reduction in energy is found, thus ensuring that a fixed point is actually a local minimum of the energy. If we use global updating we just have to hope that $\|\mathbf{x_{k+1}} - \mathbf{x_k}\|$ is small and that the term is therefore negligible. (Of course we could switch from global to single updating if the former fails to give a decrease in energy.)

4.3. The travelling salesman problem

In optimization applications, Hopfield nets are not trained. Instead we set up an energy functional corresponding to the function to be optimized, together with penalty functions for any constraints, and 'hard-wire' the weight matrix

W so that this energy is actually (4.6). We then simply start the net off and wait for it to reach a minimum. We illustrate this process by applying it to the travelling salesman problem, an application first discussed by Hopfield and Tank (see Wasserman (1989, Ch. 6) for this and other applications of Hopfield nets.) Suppose we have m cities. We wish to visit each city precisely once, returning to the starting point. Any such ordering of the cities is known as a *tour*. The aim is to find the tour of minimum length. Since the problem is NP complete, we will not expect to find the optimum solution every time: we will be satisfied with a method that produces reasonably good tours reliably. Let d_{ij} denote the distance between the ith and jth cities, so $d_{ij} = d_{ji}$. To set this problem up as a Hopfield net, we allocate m neurons to each city. Thus the net will have $n = m^2$ neurons. (This may seem excessive, but a large travelling salesman problem of say 100 cities will only require 10 000 units: potentially well within the capabilities of VLSI circuits when we remember that each unit is very simple.) Think of the neurons being arranged in a table of m rows of m units, each row corresponding to a city. Now a 1 in the jth neuron of any row means that that city will be visited jth. Thus for a valid tour, there must be exactly one 1 in each row. Since the cities are visited sequentially (we cannot be in two places at once), there must also be exactly one 1 in each column. So in total there will be m 1s in a valid tour table. To construct our energy functional we introduce penalty functions for each of these constraints. Let the state vector of the net be $\mathbf{x}$, allocated by rows of the table so that the first m elements refer to city 1, the second m elements refer to city 2 etc. Recall that $\mathbf{x}$ may contain only 0s or 1s. Denote by $\mathbf{e}$ the vector containing all 1s. We first introduce a term

$$(\mathbf{e}^T\mathbf{x} - m)^2 = \mathbf{x}^T(\mathbf{e}\mathbf{e}^T)\mathbf{x} - 2m\mathbf{x} + m^2.$$

This term will vanish if and only if $\mathbf{x}$ has exactly m 1s. The m^2 term can be ignored since this is constant and we are constructing an energy functional to minimize. The matrix $\mathbf{e}\mathbf{e}^T$, all of whose entries are 1, presents a difficulty, as it does not have zero diagonal. However, let $A = \mathbf{e}\mathbf{e}^T - I$. Then $\mathbf{x}^T(\mathbf{e}\mathbf{e}^T)\mathbf{x} = \mathbf{x}^T A\mathbf{x} + \mathbf{x}^T\mathbf{x}$. Moreover, since the entries of $\mathbf{x}$ can be only 0 or 1, we have $\mathbf{x}^T\mathbf{x} = \mathbf{e}^T\mathbf{x}$. So we can use

$$\mathbf{x}^T A\mathbf{x} + (1 - 2m)\mathbf{e}^T\mathbf{x}. \tag{4.8}$$

The other constraints are a little more complicated as we have to pick out the sections of the state vector corresponding to individual cities. This is conveniently achieved by introducing the following matrices. Let B be defined as follows: $b_{ii} = 0$, $i = 1, \ldots, n$. Also $b_{ij} = 0$ if $(\mathbf{x})_i$ and $(\mathbf{x})_j$ represent different cities or, in other words, if $[i/m] \neq [j/m]$, where $[r]$ denote the greatest integer *strictly* less than r. Finally $b_{ij} = 1$ if $(\mathbf{x})_i$ and $(\mathbf{x})_j$ represent the same city or, in other words, if $[i/m] = [j/m]$ but $i \neq j$.

The matrix B is thus symmetric and block diagonal. Each block on the diagonal corresponds to a particular city. The quadratic form

$$\mathbf{x}^T B\mathbf{x} \tag{4.9}$$

therefore has the following effect. Each block of B multiplies together and sums distinct entries of $\mathbf{x}$ corresponding to the *same* city, which will be greater than zero if two elements corresponding to the same city are nonzero. Finally these sums are themselves summed over the cities. (Note: if this is not clear, it helps to write out (4.9) for, say, $m = 3$.) Remembering that $\mathbf{x}$ can contain only 1s and 0s. we see that (4.9) will achieve its minimum value of 0 if each row of our unit table has at most one 1, i.e. if city is visited at most once. Similarly we need to ensure that we do not try and visit two cities at once, i.e. that each column of our table of units has at most one 1. This is encoded by

$$\mathbf{x}^T C\mathbf{x}, \tag{4.10}$$

where $c_{ii} = 0$, $c_{ij} = 1$ if i modulo $m = j$ modulo m but $i \neq j$, $c_{ij} = 0$ otherwise. Finally we need a term which actually reduces the length of tours. Basically, this simply requires us to read off the tour order from our table and add up the distances, but of course we need to write this process as a symmetric quadratic form. For each city, we have a 1 in the position of the table telling us when we visit it. The 1 in the previous column tells us where we got there from. Since there can be at most one 1 in this previous column, we may as well multiply the corresponding distance by the element of $\mathbf{x}$ corresponding to that entry, multiply that by the element of $\mathbf{x}$ corresponding to the current city, and sum. In order to preserve symmetry, it is best to look in both the previous and next columns. This results in each leg of the tour being included twice, but remember that we are not actually going to calculate it. We are merely using it to get the weights for our network. Of course, we also need to include the return to the starting city. The quadratic form is thus

$$\mathbf{x}^T F\mathbf{x}, \tag{4.11}$$

where we think of F as being made up of m^2 $m \times m$ blocks. The p,qth block corresponds to a trip from the pth city to the qth. Let F_{pq} be this $m \times m$ submatrix. Then $F_{pp} = 0$. For $p \neq q$, each entry in F_{pq} will be either d_{pq} or 0. The nonzero elements will occur only if the trip from city p to city q is actually made. Thus we need to put d_{pq} on the sub and super diagonals of F_{pq} . But we must also allow for the possibility that p is the first city and q the last, or *vice versa*. Therefore we must also set $(F_{pq})_{1m} = (F_{pq})_{m1} = d_{pq}$. The other elements of F_{pq} are 0.

Now choose parameters $\alpha \gg \beta, \gamma \gg \theta$. Comparing (4.6) with (4.8)–(4.11) we may set $W = \alpha A + \beta B + \gamma C + \theta F$ and $q = (1 - 2m)\mathbf{e}^T$. The network

will first try to find a tour with n 1s, causing the (4.8) term to vanish. It will then try to find a 1 in each row and column, eliminating the (4.9) and (4.10) terms. Finally it will try to reduce the length of the tour. We need to set the thresholds to (say) $\frac{1}{2}$. In fact Hopfield and Tank used not this step function activation, but $\frac{1}{2}(1+\tanh(\mathbf{x}/\mathbf{u}_0)$ (Wasserman, 1989, p. 109) where $\mathbf{x}$ is the total input to the unit and $\mathbf{u}_0$ is a further parameter. The behaviour is reportedly very dependent on the choice of parameters. Van den Bout and Miller (1988) discuss this in detail and suggest improvements. A detailed analysis of when the method will give a valid tour, and how bounds on the optimal solution may be obtained, has been provided by Aiyer *et al.* (1989) and in two papers, one by the same authors and the other by de Carvalho and Barbosa in INNC 90 (1990, pp. 245–248 and pp. 249–253, respectively). This proceedings also contains several other related papers in its section on optimization (pp. 245–297).

5. Concluding remarks

Neural networks represent an important new tool for computation. Numerical analysts can and have made significant contributions to the field. As we have seen, the topic can stimulate new research in approximation theory and optimization algorithms. Equally, we saw in Section 4 that methods of neural computation can provide new tools for numerical computation. In particular, it seems plausible that the Hopfield net could be applied to processor assignment problems in parallel computation, particularly since, by definition, suitable parallel hardware would be available to implement it! There are other issues in neural net research which have hardly been treated formally at all. For example, the claim is often made that networks are fault tolerant in that deleting the connection to a unit, or even a whole unit, degrades the performance only marginally rather than catastrophically as it would with a conventional computing system. The argument for this is that networks store information in a distributed way, so only a little information will be lost. This is certainly the case if the network is constructed as in Section 2.2, as deleting one of the terms from the quadrature formulae will not destroy convergence. But networks are not constructed this way in practice. Little serious analysis of fault tolerance has been attempted. Applying numerical analysis to neural networks will not only be useful in applications, but should provide a new stimulus to numerical analysis itself.

Acknowledgement

I would like to thank Professor W. Light, of the University of Leicester, for some useful discussions on the material in Section 2.2.

REFERENCES

I. Aleksander and H. Morton (1990), *An Introduction to Neural Computing*, Chapman and Hall (London).

L. B. Almeida and F. M. Silva (1992), 'Adaptive decorrelation', in *Artificial Neural Networks 2* (I. Aleksander and J. Taylor, eds), Vol. 2, North-Holland (Amsterdam) 149–156.

S. I. Amari (1990), 'Mathematical foundations of neurocomputing', *Proc. IEEE* **78**, 1143–1463.

S. V. B. Aiyer, M. Niranjan and F. Fallside (1989), 'A theoretical investigation into the performance of the Hopfield model', *Tech. Report, CUED/F-INFENG/TR 36*, Cambridge University Engineering Department, Cambridge, CB2 1PZ, England.

P. Baldi and K. Hornik (1989), 'Neural networks and principal component analysis: learning from examples without local minima', *Neural Networks* **2**, 53–58.

A. Ben-Israel and T. N. E. Greville (1974), *Generalised Inverses, Theory and Applications*, Wiley (Chichester).

M. Bichsel and P. Seitz (1989), 'Minimum class entropy: a maximum information approach to layered networks', *Neural Networks* **2**, 133–141.

R. W. Brause (1992), 'The error bounded descriptional complexity of approximation networks', *Fachberiech Informatik*, J W Goethe University, Frankfurt am Main, Germany.

D. S. Broomhead and D. Lowe (1988), 'Multivariable function interpolation and adaptive networks', *Complex Systems* **2**, 321–355.

T. Chen, H. Chen and R. Liu (1991), 'A constructive proof and extension of Cybenko's approximation theorem', in *Computing Science and Statistics: Proc. 22nd Symp. on the Interface*, Springer (Berlin) 163–168.

E. W. Cheney (1966), *Introduction to Approximation Theory*, McGraw-Hill (New York).

G. Cybenko (1989), '∞ approximation by superpositions of a sigmoidal function', *Math. Control-Signals Systems* **2**, 303–314.

P. Diaconis and M. Shashahani (1984), 'On nonlinear functions of linear combinations', *SIAM J. Sci. Statist. Comput.* **5**, 175–191.

S. W. Ellacott (1990), 'An analysis of the delta rule', *Proc. Int. Neural Net Conf., Paris* Kluwer (Deventer) 956–959.

S. W. Ellacott (1993a), 'The numerical analysis approach', in *Mathematical Approaches to Neural Networks* (J.G. Taylor, ed.), North Holland (Amsterdam) 103–138.

S. W. Ellacott (1993b), 'Techniques for the mathematical analysis of neural networks', *J. Appl. Comput. Math.* to appear.

S. W. Ellacott (1993c), 'Singular values and neural network algorithms', in *Proc. British Neural Network Society Meeting, February 1993*.

K. Falconer (1990), *Fractal Geometry*, Wiley (New York).

M. Fombellida and J. Destiné (1992), 'The extended quickprop', in *Artificial Neural Networks 2* (I. Aleksander and J. Taylor), Vol. 2, North-Holland (Amsterdam) 973–977.

K.-I. Funahashi (1989), 'On the approximate realization of continuous mappings by neural networks', *Neural Networks* **2**, 183–192.

K. Hornik K, M. Stinchcombe and H. White (1989), 'Multilayer feedforward networks are universal approximators', *Neural Networks* **2**, 359–366.

INNC 90 (1990), *Proc. Int. Neural Network Conf. (9–13 July 1990, Palais de Congres, Paris, France)* Kluwer (Deventer).

E. Isaacson and H. B. Keller (1966), *Analysis of Numerical Methods*, Wiley (New York).

D. Jacobs (ed.) (1977), *The State of the Art in Numerical Analysis*, Academic Press (New York).

A. J. Jones (1992), 'Neural computing applications to prediction and control', Department of Computing, Imperial College, London, United Kingdom.

E. Kreyszig (1978), *Introductory Functional Analysis with Applications*, Wiley (New York).

B. Linggard and C. Nightingale (eds) (1992), *Neural Networks for Images, Speech and Natural Language*, Chapman and Hall (London).

W. Light (1992), 'Ridge function, sigmoidal functions and neural networks', in *Approximation Theory VII* (E. W. Cheney, C. K. Chui and L. L. Schumaker, eds) Academic (Boston) 1–44.

J. C. Mason and P. C. Parks (1992), 'Selection of neural network structures – some approximation theory guidelines', Ch. 8, in *Neural Networks for Control and Systems*, (K. Warwick, G. W. Irwin and K. J. Hunt, eds) *IEE Control Engineering Series no. 46*, Peter Peregrinus (Letchworth).

H. N. Mhaskar and C. Micchelli (1992), 'Approximation by superposition of sigmoidal functions', *Adv. Appl. Math.* **13**, 350–373.

H. N. Mhaskar (1993), 'Approximation properties of a multilayered feedforward artificial neural network', *Adv. Comput. Math.* **1**, 61–80.

J. J. Moré (1978), 'The Levenberg–Marquardt algorithm, implementation and theory', in *Proc. Dundee Biennial Conf. on Numerical Analysis 1977*, (G. A. Watson, ed.), *Springer Lecture Notes in Mathematics no. 630*, Springer (Berlin) 105–116.

E. Oja (1983), *Subspace Methods of Pattern Recognition*, Research Studies Press (Letchworth, UK).

E. Oja (1992), 'Principal components, minor components and linear neural networks', *Neural Networks* **5**, 927–935.

E. Oja, H. Ogawa and J. Wangviwattana (1992), 'PCA in fully parallel neural networks', in *Artificial Neural Networks 2* (I. Aleksander and J. Taylor, eds), Vol. 2, North-Holland (Amsterdam) 199–202.

M. J. D. Powell (1992), 'The theory of radial basis functions approximation in 1990', in *Advances in Numerical Analysis* (W. Light, ed.), Vol. II, Oxford University Press (Oxford), 105–210.

D. E. Rumelhart and J. L. McClelland (1986), *Parallel and Distributed Processing: Explorations in the Microstructure of Cognition*, Vols 1 and 2, MIT (Cambridge, MA).

H. Sagan (1969), *Introduction to the Calculus of Variations*, McGraw-Hill (New York).

P. K. Simpson (1990), *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations*, Pergamon Press (New York).

E. Stein and G. Weiss (1971), *Introduction to Fourier Analysis on Euclidean Spaces*, Princeton University Press (Princeton, USA).

J.G. Taylor (ed.) (1993), *Mathematical Approaches to Neural Networks*, North Holland (Amsterdam).

D. E. Van den Bout and T. K. Miller (1988), 'A travelling salesman objective function that works', *Proc. IEEE Conf. on Neural Networks*, Vol. 2, SOS Printing (San Diego, CA) 299–304.

G. Venkataraman and G. Athithan G (1991), 'Spin glass, the travelling salesman problem, neural networks and all that', *Prāmana J. Phys.* **36**, 1–77.

Z. Wang, M. T. Tham and A. J. Morris (1992), 'Multilayer feedforward neural networks: a cannonical form approximation of nonlinearity', Department of Chemical and Process Engineering, University of Newcastle upon Tyne, Newcastle upon Tyne, NE1 7RU, United Kingdom.

K. Warwick, G. W. Irwin and K. J. Hunt (1992), 'Neural networks for control and systems', *IEE Control Engineering Series No. 46*, Peter Peregrinus (Letchworth).

P. D. Wasserman (1989), *Neural Computing: Theory and Practice*, Van Nostrand Reinhold (New York).

P. J. Werbos (1992), 'Neurocontrol: where it is going and why it is crucial', in *Artificial Neural Networks 2* (I. Aleksander and J. Taylor, eds), Vol. 1, North-Holland (Amsterdam) 61–68.

Y. Xu, W. A. Light and E. W. Cheney (1991), 'Constructive methods of approximation by ridge functions and radial functions'. Address of first author: Department of Mathematics, University of Arkansas at Little Rock, Little Rock, AR 72204, USA.