ELSEVIER

Journal of Computational and Applied Mathematics 50 (1994) 283–297

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Techniques for the mathematical analysis of neural networks

S.W. Ellacott

Department of Mathematical Sciences, University of Brighton, Moulsecoomb, Brighton BN2 4GJ, United Kingdom

Received 27 July 1992; revised 26 November 1992

Abstract

This expository paper covers the following topics: (1) a very brief introduction to neural networks for those unfamiliar with the basic concepts; (2) an equally brief survey of various mathematical approaches to neural systems with an emphasis on approximation theory; (3) an algorithmic approach to the analysis of networks developed by this author using the tools of numerical linear algebra. This approach is novel and was first proposed by the author in (1990).

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, with a new result on the effect of linear filtering on the rate of convergence of the delta rule.

Key words: Neural networks; Numerical linear algebra; Numerical analysis

1. An introduction to neural networks

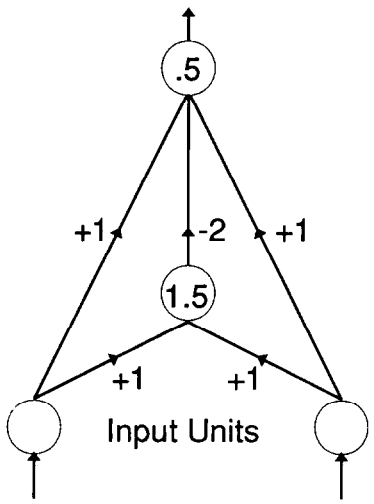
1.1. A network to compute "xor"

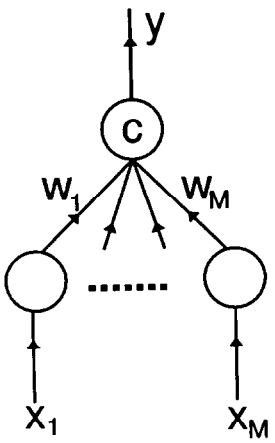
A neural network is a model of computation based loosely on the mammalian brain. Rather than give a formal definition, we illustrate by a simple example. Fig. 1.1 shows a network designed to compute the "exclusive-or" function. Each input unit takes a single scalar input. In general these may take any 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

SSDI 0377-0427(93)E0149-G





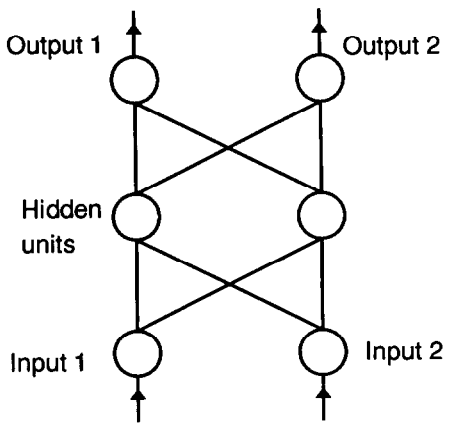


Fig. 1.1.
Fig. 1.2.
Fig. 1.3.

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 inscribed ".5" is thus $1 \times 1 - 2 \times 0 + 1 \times 0 = 1$. Since $1 > 0.5$, 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.

1.2. Perceptrons and multilayer perceptrons

There is a vast range of variations on this general idea: the reader should consult one of the many textbooks available on this subject. For an introductory treatment, see [1,14,17]. A deeper work, although now a little dated, is [13]. In this paper we shall concentrate on the simplest of all neural networks, the *perceptron*, and an extension of this called the *multilayer perceptron* (MLP) or *semilinear feedforward network*. The latter is the most popular of all neural network architectures, probably because it is relatively easy to understand. However, many of the ideas are much more widely applicable and the formulation of the backpropagation algorithm given in Section 3.3 is certainly much more general than is required just for the MLP.

Fig. 1.2 shows a simple perceptron with a single output. The units are interpreted as in Fig. 1.1 with the input units having a identity activation function and the output unit having a simple threshold. Thus denoting the input vector $\boldsymbol{x}$ and the weight vector by $\boldsymbol{w}$ (both in $\mathbb{R}^M$), it is easy

to see that the output is 1 if $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} > c$ and 0 if $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \leqslant c$. So for a fixed weight vector $\boldsymbol{w}$ and threshold c, the network divides $\mathbb{R}^M$ into two half spaces separated by a hyperplane. For obvious reasons, the perceptron is described as a linear network. Considering the case $M = 2$, 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 may years until tools for handling nonlinear networks such as Fig. 1.1 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 MLP shown in Fig. 1.3 is the most obvious and straightforward generalization. It consists simply of layers of perceptrons connected together in cascade. In the diagram 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.

2. Analysis of neural networks

In this section we give a brief survey of some of the mathematical techniques that have been used to analyse neural networks. In order to prevent the bibliography becoming inordinately long, we shall as far as possible make reference to survey articles rather than original source material. Thus we attempt a survey of surveys!

2.1. Classification and approximation properties

A natural question arising from the ideas developed in Section 1 is to consider what sets of points a given network can classify. In fact it is not hard to see that an MLP can separate any finite sets of points in $\mathbb{R}^M$. For let A and B be two such sets. 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 $P_1, \ldots, P_k$ such that

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

Each P_j consists of a finite intersection of half spaces. It can thus be obtained by a network computing the "and" function which is linearly separable. The union to include A can then be obtained by a network computing the "or" function: 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 $\mathbb{R}^M$. The output $\boldsymbol{y}$ of the network is a vector in $\mathbb{R}^N$ where usually $N \ll M$. (In many cases $N = 1$.) The network thus computes a function $\boldsymbol{G}: \mathbb{R}^M \to \mathbb{R}^N$ which we regard as an *approximation* to some other function $\boldsymbol{H}: \mathbb{R}^M \to \mathbb{R}^N$. The

point classification problem discussed above can be put into this context by choosing $N = 1$ and $\boldsymbol{H}$ to be the characteristic function of the set Q in (2.1) (i.e., $\boldsymbol{H}(x) = 1$ 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 $\boldsymbol{G}$. (Sometimes networks are actually constructed this way: radial basis function networks are of this type [10]. As constructed here, the function $\boldsymbol{G}$ 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 of possible functions $\boldsymbol{G}$ corresponding to our particular class of network is dense in some suitable function space which includes our target function $\boldsymbol{H}$. Let us pursue this idea a little further.

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 $f: \mathbb{R} \to \mathbb{R}$. Desirable activation functions should have the property of being monotonic increasing, bounded and *sigmoidal*, which means that simply the limits at $\pm\infty$ are 0 and 1, respectively. The most popular choice in practice is

$$f(x) = \frac{1}{1 + \exp(-x)}. \tag{2.2}$$

Proposition 2.1. *If the input vectors are restricted to a compact subset $C \subset \mathbb{R}^M$, then any continuous function $\boldsymbol{H}: C \to \mathbb{R}$ can be approximated with arbitrary small uniform error by a multilayer perceptron in which the input and output layers have identity activation functions, and there is a* single *hidden layer of units for which the activation function is* (2.2).

In fact the density theorems will work for activation functions chosen from much more general classes of monotonic bounded sigmoidal functions, but with certain technical restrictions. The first result of this type appears to have been proved by Cybenko in 1989, but more accessible approaches have followed. See [10,11,18] for recent work. From the practical viewpoint, however, mere density is of limited usefulness. We need to know (or at least be able to estimate) how many hidden units are required to give a particular error. In classical approximation theory, results of this type are called Jackson Theorems. Reference [11] has some results of this type, although they are not yet sufficiently powerful to be really useful.

The density approach 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 of sixteen bits. There may be no point in using a very accurate network if its realisation introduces large errors. This problem has been addressed in [4].

2.2. Dynamic behaviour

Up to now we have considered networks as essentially static entities, unchanging over time. However, there are many ways in which dynamic ideas can be introduced. For the simple

multilayer perceptrons discussed so far, we need to consider how the weights are chosen via *learning algorithms*. This topic forms the subject of Section 3. However, it is worth noting at this point that dynamic behaviour can be introduced into networks in many different ways. Some network architectures are recursive (consider the output of an MLP being fed back in as part of the input). Others are defined by or approximated by differential equations. Some authors, particularly those interested in modelling biological systems, have considered architectures involving coupled oscillators and/or chaos theory [9]. In fact the whole panoply of dynamical systems and control theory underlies the study of neural networks in a manner too pervasive to be adequately surveyed here. Equally, insights from statistical physics have been introduced [15]. Differential topology has its adherents [16]. The structure of the classification space can be analysed using statistical decision theory [2]. Indeed the theory of neural networks would appear to be almost as chaotic as their dynamic behaviour. It seems certain that 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 disciplines. Notwithstanding this remark, we will now proceed to investigate how machine learning can be considered as a numerical algorithm!

3. Numerical analysis of learning algorithms

3.1. The delta rule

We begin by considering the simplest of all neural models, the simple perceptron. Fig. 1.2 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. Denote the training vectors (generically) by $\boldsymbol{x}$ and *desired* output vectors by $\boldsymbol{y}$. As in Section 2, 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 $\boldsymbol{y}$ sufficiently well by the network, then obviously a suitable choice of c will solve the classification problem. W is the weight matrix. In summary, then, we wish to find W such that $\boldsymbol{y} \approx W\boldsymbol{x}$ for all pairs $(\boldsymbol{x}, \boldsymbol{y})$ 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 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t\}$ of input patterns and for each $\boldsymbol{x}_i$ we supply the corresponding output $\boldsymbol{y}_i$. The system uses these pattern pairs to update its estimate of the desired weight matrix W. At heart, a learning algorithm is thus simply an optimisation process, but it has the special feature that the patterns are supplied serially rather than simultaneously as in standard least-squares approximation and optimisation.

We assume initially that W is updated after each training pattern. The change in W when the pattern $\boldsymbol{x}$ is presented is given by [13, p.332]

$$(\delta W)_{ji} = \eta\big(\boldsymbol{y}_j - (W\boldsymbol{x})_j\big)x_i,$$

where η is a parameter to be chosen, called the *learning rate*, and $(W\boldsymbol{x})_j$ denotes the jth element of $W\boldsymbol{x}$. Thus, if the error term in brackets is (say) positive, we will add a component of

x to each row of W, increasing the output of the network for this pattern. Actually, 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 w^{T}. Hence *without loss of generality* we return to the single-output perceptron (Fig. 1.2). We get

$$\delta w_i = \eta(y - w^{\mathrm{T}}x)x_i, \quad \text{so } \delta w = \eta(y - w^{\mathrm{T}}x)x.$$

Thus given a current iterate weight vector w_k,

$$w_{k+1} = w_k + \delta w_k = w_k + \eta(y - w_x^{\mathrm{T}}x)x = (I - \eta xx^{\mathrm{T}})w_k + \eta yx. \tag{3.1}$$

The final equation is obtained by transposing the (scalar) quantity in brackets. Note the subscript 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 x to the current weight vector. It is usual to analyse this iteration in the asymptotic case as $\eta \to 0$, but it is not used this way in practice, so it is more relevant to consider a fixed η [5]. We now prove some results about this iteration: the first lemma is a special case of a well-known result (see, e.g., [12, p.18]). The proof is a direct verification.

Lemma 3.1. *Let* $B = (I - \eta xx^{\mathrm{T}})$. *Then* B *has only two distinct eigenvalues*: $1 - \eta \| x \|_2^2$ *corresponding to the eigenvector* x *and* 1 *corresponding to the subspace of vectors orthogonal to* x. (*Here* $\| \cdot \|_2$ *denotes the usual Euclidean norm.*)

As an immediate consequence (see [7, p.10, Eq. (11)], we obtain the following lemma.

Lemma 3.2. *Provided* $0 \leqslant \eta \leqslant 2/\| x \|_2^2$, *we have* $\| B \|_2 = \rho(B) = 1$, *where* $\rho(B)$ *is the spectral radius of* B.

Now suppose we actually have t pattern vectors $x_1, \ldots, x_t$. We will assume temporarily that these span the space of input vectors, i.e., if the x's are M-vectors, then the set of pattern vectors contains M linearly independent ones. (This restriction will be removed later.)

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

Lemma 3.3. *If* $0 < \eta < 2/\| x_p \|_2^2$ *holds for each training pattern* x_p, *and if the* x_p *span, then* $\| A \|_2 < 1$.

Proof. By definition, there exists v such that $\| A \|_2 = \| Av \|_2$ and $\| v \|_2 = 1$. Thus $\| A_2 \| = \| B_t B_{t-1} \cdots B_1 v \|_2 \leqslant \| B_t B_{t-1} \cdots B_2 \|_2 \| B_1 v \|_2$ from the definition of the norm). We identify two cases.

(Case 1) If $v^{\mathrm{T}}x_1 \neq 0$, $\| B_1 v \|_2 < 1$, since the component of v in the direction of x is reduced (see Lemma 3.1: if this is not clear, write v in terms of x and the perpendicular component, and apply B_1 to it). On the other hand, $\| B_t B_{t-1} \cdots B_2 \|_2 \leqslant \| B_t \|_2 \| B_{t-1} \|_2 \cdots \| B_2 \|_2 = 1$.

(Case 2) If $v^{\mathrm{T}}x_1 = 0$, then $B_1 v = v$ (Lemma 3.1). Hence, $\| A \|_2 = \| B_t B_{t-1} \cdots B_2 v \|_2$ and we may carry on removing B's until Case 1 applies. □

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

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

where Λ is as defined above and

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

Here, of course, y_p denotes the target y-value for the pth pattern, not the pth element of a vector. Note that the B's and hence h depend on η and the x's, but *not* on the current 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 x_p span the space of input vectors, then for sufficiently small η, $\| \Lambda \|_2 < 1$. Hence the mapping $F(w) = \Lambda w + \eta h$ satisfies

$$\| F(w) - F(v) \|_2 = \| \Lambda(w - v) \|_2 \leqslant \| \Lambda \|_2 \| w - v \|_2, \tag{3.3}$$

i.e., it is contractive with contraction parameter $\| \Lambda \|_2$. It follows from the Contraction Mapping Theorem that the iteration (3.2a) does have a fixed point. The theorem also guarantees that the fixed point is unique. Now if there exists a w that makes all the errors zero, then it is easy to verify that this 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 $w(\eta)$. In the limit, as the iteration (3.1) runs through the patterns, it will generate a limit cycle of vectors w_k returning to $w(\eta)$ after the cycle of t patterns has been completed.

Again assuming that the x_p span, it can be shown using standard results on the continuity of solutions of linear equations that as $\eta \to 0$, $w(\eta)$ tends to the weight vector w, which gives the *least-squares* error over the patterns [5,6]. Unfortunately the rate of convergence is proportional to the norm of the autocorrelation matrix occurring in the normal equations, and, as we shall see in the next section, this can be large.

Finally, we need to consider what happens when the 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.

3.2. The "epoch method"

Since we are assuming that we have a fixed and finite set of patterns x_p, $p = 1, \ldots, t$, 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 it can be shown 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

$$w_{k+1} = w_k - \eta \sum_{p=1}^{t} \left(x_p x_p^{\mathrm{T}}\right) w_k + \eta \sum_{p=1}^{t} (y_p x_p) = \Omega w_k - \eta \sum_{p=1}^{t} (y_p x_p), \tag{3.4}$$

where $\Omega = (I - \eta X X^{\mathrm{T}}) = (I - \eta L)$, say. (Here X is the matrix whose columns are the X_p's.)

Formula (3.4) 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. Unfortunately, however, there is a catch! To see what this is, we need to examine the eigenvalues of Ω.

Clearly $L = XX^{\mathrm{T}}$ is symmetric and positive semi-definite. Thus it has real nonnegative eigenvalues. In fact, provided the $\boldsymbol{x}_p$ span, it is (as is well known) strictly positive definite.

The eigenvalues of Ω are $1 - \eta$(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, $\rho(\Omega) = \| \Omega \|_2 < 1$.

Hence the iteration (3.4) 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 $L = XX^{\mathrm{T}}$ has eigenvalues λ_j, $j = 1, \ldots, M$, with

$$0 \leqslant \lambda_M \leqslant \lambda_{M-1} \leqslant \cdots \leqslant \lambda_1 = \rho(XX^{\mathrm{T}}) = \| XX^{\mathrm{T}} \|_2 = \| X^{\mathrm{T}} \|_2^2.$$

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

$$\lambda = \| X^{\mathrm{T}} \|_2^2 = \max_{\| \boldsymbol{v} \|_2 = 1} \| X^{\mathrm{T}} \boldsymbol{v} \|_2^2 = \max_{\| \boldsymbol{v} \|_2 = 1} \boldsymbol{v}^{\mathrm{T}} XX^{\mathrm{T}} \boldsymbol{v} \tag{3.5}$$

$$\leqslant \sum_{p=1}^{t} \| \boldsymbol{x}_p \|_2^2.$$

On the other hand, we can get a lower bound by substituting a particular $\boldsymbol{v}$ into the expression on the right-hand side of (3.5). For instance, we have for any k, $k = 1, \ldots, t$,

$$\lambda_i \geqslant \frac{1}{\| \boldsymbol{x}_k \|_2} \left(\sum_{p=1}^{t} \left(\boldsymbol{x}_k^{\mathrm{T}} \boldsymbol{x}_p \right)^2 \right) \geqslant \| \boldsymbol{x}_k \|_2. \tag{3.6}$$

Now consider a particular case. Suppose the $\boldsymbol{x}_p$ cluster around two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ which are mutually ortho*normal*. If these represent two classes which are to be separated, we are in the ideal situation for machine learning. 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 (3.6)

$$\lim_{\epsilon \to 0} \lambda_1 \geqslant \tfrac{1}{2} t,$$

and since the rank of $L = XX^{\mathrm{T}}$ collapses to 2,

$$\lim_{\epsilon \to 0} \lambda_M = 0.$$

Thus, unlike the ordinary delta rule for which the convergence condition depends only on the norm of the individual patterns, for the epoch method we may require an arbitrary small η to get convergence.

3.3. Generalisation 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 generalise to nonlinear systems such as the backpropagation algorithm for the MLP. 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 generalisation 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: M in our previous notation. Now we will let M denote the *total number* of weights and n the input dimension.

So the input patterns $\boldsymbol{x}$ to our network are in $\mathbb{R}^n$, and we have a vector $\boldsymbol{w}$ 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" $\boldsymbol{w}$ is the $m \times n$ weight matrix, and thus $M = mn$. For a multilayer perceptron, $\boldsymbol{w}$ is the Cartesian product of the weight matrices in each layer. For a general system with m outputs, the network computes a function $\boldsymbol{G} : \mathbb{R}^M \times \mathbb{R}^n \to \mathbb{R}^m$. Say

$$\boldsymbol{v} = \boldsymbol{G}(\boldsymbol{w}, \boldsymbol{x}),$$

where $\boldsymbol{v} \in \mathbb{R}^m$. We equip $\mathbb{R}^M$, $\mathbb{R}^m$ and $\mathbb{R}^n$ with suitable norms $\|\cdot\|$. Since these spaces are finite-dimensional, it does not really matter which norms are adopted, but it is convenient to use the Euclidean norm $\|\cdot\|_2$. For pattern $\boldsymbol{x}_p$, denote the corresponding output by $\boldsymbol{v}_p$, i.e.,

$$\boldsymbol{v}_p = \boldsymbol{G}(\boldsymbol{w}, \boldsymbol{x}_p).$$

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

$$\boldsymbol{G}(\boldsymbol{w} + \delta\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{G}(\boldsymbol{w}, \boldsymbol{x}) + D(\boldsymbol{w}, \boldsymbol{x})\delta\boldsymbol{w} + \mathrm{o}(\|\delta\boldsymbol{w}\|). \tag{3.7}$$

On the other hand, for given $\boldsymbol{w}$, corresponding to a particular pattern $\boldsymbol{x}_p$, we have a desired output $\boldsymbol{y}_p$ and thus an error ϵ_p given by

$$\epsilon_p^2 = (\boldsymbol{y}_p - \boldsymbol{v}_p)^{\mathrm{T}}(\boldsymbol{y}_p - \boldsymbol{v}_p) = \boldsymbol{q}_p^{\mathrm{T}}\boldsymbol{q}_p, \quad \text{say.} \tag{3.8}$$

The total error is obtained by summing the ϵ_p^2's over the t available patterns, thus

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

An ordinary descent algorithm will seek to minimise 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 $\delta \boldsymbol{q}_p$ in $\boldsymbol{q}_p$ we have from (3.8)

$$\delta\epsilon_p^2 = (\boldsymbol{q}_p + \delta\boldsymbol{q}_p)^{\mathrm{T}}(\boldsymbol{q}_p + \delta\boldsymbol{q}_p) - \boldsymbol{q}_p^{\mathrm{T}}\boldsymbol{q}_p = 2\delta\boldsymbol{q}_{pp}^{\mathrm{T}} + \delta\boldsymbol{q}_p^{\mathrm{T}}\delta\boldsymbol{q}_p.$$

Since $\boldsymbol{y}_p$ is fixed,

$$\delta\boldsymbol{q}_p = -\delta\boldsymbol{v}_p = -D(\boldsymbol{w}, \boldsymbol{x}_p)\delta\boldsymbol{w} + \mathrm{o}(\|\delta\boldsymbol{w}\|), \quad \text{by (3.7)}.$$

Thus,

$$\begin{aligned}\delta\epsilon_p^2 &= -2\big(D(\boldsymbol{w}, \boldsymbol{x}_p)\delta\boldsymbol{w}\big)^{\mathrm{T}}\big(\boldsymbol{y}_p - \boldsymbol{G}(\boldsymbol{w}, \boldsymbol{x}_p)\big) + \mathrm{o}(\|\delta\boldsymbol{w}\|)\\ &= -2\delta\boldsymbol{w}^{\mathrm{T}}\big(D(\boldsymbol{w}, \boldsymbol{x}_p)\big)^{\mathrm{T}}\big(\boldsymbol{y}_p - \boldsymbol{G}(\boldsymbol{w}, \boldsymbol{x}_p)\big) + \mathrm{o}(\|\delta\boldsymbol{w}\|).\end{aligned}$$

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

$$\delta\boldsymbol{w} = \eta\big(D(\boldsymbol{w}, \boldsymbol{x}_p)\big)^{\mathrm{T}}\big(\boldsymbol{y}_p - \boldsymbol{G}(\boldsymbol{w}, \boldsymbol{x}_p)\big).$$

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

$$\boldsymbol{G}(\boldsymbol{w}, \boldsymbol{x}_p) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_p,$$

and the derivative is the gradient vector (considered as a row vector) obtained by differentiating this with respect to $\boldsymbol{w}$, i.e. $\boldsymbol{x}_p^{\mathrm{T}}$. Thus we indeed have a generalisation of (3.1). Given a kth weight vector $\boldsymbol{w}_k$, we have

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \delta\boldsymbol{w}_k = \boldsymbol{w}_k + \eta\big(D(\boldsymbol{w}_k, \boldsymbol{x}_p)\big)^{\mathrm{T}}\big(\boldsymbol{y}_p - \boldsymbol{G}(\boldsymbol{w}_k, \boldsymbol{x}_p)\big). \tag{3.9}$$

The backpropagation rule [13, pp. 322–328] used in many neural net applications is a special case of this.

To proceed further, we need to make evident the connection between (3.9) 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 $\boldsymbol{w}^*$, of the least-square 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 (3.8) and (3.9) we obtain (assuming continuity and uniform boundedness of D in a neighbourhood of $\boldsymbol{w}^*$)

$$\begin{aligned}\boldsymbol{w}_{k+1} &= \boldsymbol{w}_k + \eta\big(D(\boldsymbol{w}_k, \boldsymbol{x}_p)\big)^{\mathrm{T}}\big(\boldsymbol{y}_p - \boldsymbol{G}(\boldsymbol{w}^*, \boldsymbol{x}_p) - D(\boldsymbol{w}^*, \boldsymbol{x}_p)(\boldsymbol{w}_k - \boldsymbol{w}^*)\big) + \mathrm{o}(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|)\\ &= \big(I - \eta D(\boldsymbol{w}_k, \boldsymbol{x}_p)^{\mathrm{T}}D(\boldsymbol{w}^*, \boldsymbol{x}_p)\big)\boldsymbol{w}_k + \eta\big(D(\boldsymbol{w}_k, \boldsymbol{x}_p)\big)^{\mathrm{T}}\\ &\quad\times\big(\boldsymbol{y}_p - \boldsymbol{G}(\boldsymbol{w}^*, \boldsymbol{x}_p) + D(\boldsymbol{w}^*, \boldsymbol{x}_p)\boldsymbol{w}^*\big) + \mathrm{o}(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|).\end{aligned} \tag{3.10}$$

The connection between (3.9) and (3.1) is now clear. Observe that the iteration matrix $(I - \eta D(\boldsymbol{w}_k, \boldsymbol{x}_p)^{\mathrm{T}}D(\boldsymbol{w}^*, \boldsymbol{x}_p))$ is not exactly symmetric in this case, although it will be nearly so if

w_k is close to w^*. More precisely, let us assume that $D(w, x)$ is Lipschitz continuous at w^*, uniformly over the space of pattern vectors x. Then we have

$$w_{k+1} = \big(I - \eta D(w^*, x_p)^{\mathrm{T}} D(w^*, x_p)\big) w_k + \eta \big(D(w^*, x_p)\big)^{\mathrm{T}} \times \big(y_p - G(w^*, x_p) + D(w^*, x_p) w^*\big) + \mathrm{O}(\| w_k - w^* \|). \tag{3.11}$$

Suppose we apply the patterns $x_1, \ldots, x_t$ cyclically, as for the linear case. If we can prove that the linearised part (i.e., what we would get if we applied (3.11) without O-term) of the mapping $w_k \to w_{k+t}$ is contractive, it will follow by continuity that there is a neighbourhood of 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 $D(w^*, x_p)^{\mathrm{T}} D(w^*, x_p)$ is positive semi-definite. Thus for η sufficiently small, $\| I - \eta D(w^*, x_p)^{\mathrm{T}} D(w^*, x_p) \|_2 \leqslant 1$. 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

$$\eta < \frac{1}{\rho\big(D(w^*, x_p)^{\mathrm{T}} D(w^*, x_p)\big)}.$$

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 linearised product mapping will have norm 1, so the nonlinear map could actually be expansive on some subspace. For brevity, we will not pursue this detail here.

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 the 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 (3.4). 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. Proposition 4.1 is, to this author's knowledge, original.

4.1. Overgeneralisation

Firstly, we can provide a simple explanation for the well-known phenomenon of *overgeneralisation* 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. Recall (3.4):

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

where $\Omega = (I - \eta XX^{\mathrm{T}})$. We decompose X in singular-value form. (See, e.g., [3, Chapter 6], [8, pp. 3–53, 269–312].) Specifically we may write

$$X = PSQ^{\mathrm{T}}, \tag{4.1}$$

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

$$w_{k+1} = (I - \eta PSS^{\mathrm{T}}P^{\mathrm{T}})w_k - \eta Xy = P(I - \eta SS^{\mathrm{T}})P^{T}w_k - \eta PSQ^{\mathrm{T}}y,$$

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

$$z_{k+1} = (I - \eta SS^{\mathrm{T}})z_k - \eta Su. \tag{4.2}$$

At this point the notation becomes a little messy: let us denote by $(z_k)_i$ the ith element of z_k. These elements are decoupled by the SVD. More specifically, if X has r nonzero singular values (the diagonal elements of S) $\nu_1 \geqslant \nu_2 \geqslant \cdots \geqslant \nu_r$, (4.2) when written elementwise gives

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

and

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

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 $(z_k)_i$ 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 (4.2) 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. Hence the phenomenon of overgeneralisation. Initially the iteration picks out significant features in the variability of the data. Continued iteration makes it try to separate insignificant features or noise.

4.2. Filters

Many authors have commented on the advisability of performing some preprocessing of the input patterns before feeding them to the network. Often the preprocessing suggested is linear. At first sight this seems to be a pointless exercise, for if the raw input data vector is x, with dimension l, say, the preprocessing operation is represented by the $n \times l$ matrix T, W is the

input matrix of the net and we denote by the vector $\boldsymbol{h}$ the input to the next layer of the net, then

$$\boldsymbol{h} = WT\boldsymbol{x}. \tag{4.3}$$

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.

(i) 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.)

(ii) To improve the search geometry by removing principal components of the data and corresponding singular values that are irrelevant to the classification problem.

(iii) 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: in the language of numerical analysis to *precondition* the iteration.

We will not address all these three points here directly. 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 (3.4) that the crucial issue is the relationship between the unfiltered update matrix

$$\Omega = (I - \eta XX^{\mathrm{T}}) \tag{4.4}$$

and its filtered equivalent

$$(I - \eta TXX^{\mathrm{T}}T^{\mathrm{T}}) = \Omega', \quad \text{say.} \tag{4.5}$$

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. 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 $L = XX^{\mathrm{T}}$ has eigenvalues λ_j, $j = 1, \ldots, n$, with

$$0 < \lambda_n \leqslant \lambda_{n-1} \leqslant \cdots \leqslant \lambda_1 = \rho(XX^{\mathrm{T}}) = \| XX^{\mathrm{T}} \|_2 = \| X^{\mathrm{T}} \|_2^2.$$

(Note here we assume the $\boldsymbol{x}$'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 $TXX^{\mathrm{T}}T^{\mathrm{T}} = L'$, say. Let L' have eigenvalues $\mu_1 \geqslant \mu_2 \geqslant \cdots \geqslant \mu_{n'} > 0$ and T have singular values $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_{n'} > 0$. Note that we are assuming T has full rank n'.

Proposition 4.1. *With the notation above* $\mu_1 \leqslant \sigma_1^2 \lambda_1$ *and* $\mu_{n'} \geqslant \sigma_{n'}^2 \lambda_n$.

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

$$\mu_1 = \| TXX^{\mathrm{T}}T^{\mathrm{T}} \|_2 \leqslant \| T \|_2 \| XX^{\mathrm{T}} \|_2 \| T^{\mathrm{T}} \|_2 = \sigma_1^2 \lambda_1.$$

The second inequality is slightly more difficult. Let $\boldsymbol{u}_n$ be the normalised eigenvector of L' corresponding to μ_n. Then,

$$\mu_{n'} = \boldsymbol{u}_n^{\mathrm{T}} \mu_n \boldsymbol{u}_n = \boldsymbol{u}_n T^{\mathrm{T}} XX^{\mathrm{T}} T^{\mathrm{T}} \boldsymbol{u}_n = \| X^{\mathrm{T}} T^{\mathrm{T}} \boldsymbol{u}_n \|_2^2.$$

But $\| X^{\mathrm{T}} T^{\mathrm{T}} \boldsymbol{u}_n \|_2 \geqslant \lambda_n^{1/2} \sigma_{n'}$, as may be found by writing both matrices in terms of their singular-value decompositions. □

This result means that $\| \Omega' \|_2$ cannot be much larger than $\| \Omega \|_2$ if T has singular values close to 1.

Corollary 4.2. *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 [6]). 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 orthogonal complement). It follows from Proposition 4.1 that the norm of Ω' in (4.5) cannot be greater than that of Ω in (4.4). □

We observe in passing that the singular-value decomposition can also be used to study the problem of preconditioning the iteration [6].

5. Concluding remark

Neural networks are studied not with the discrete mathematical tools of theoretical computer science, but with the classical methods of dynamical systems, approximation theory, statistical physics and numerical analysis. They are likely to prove a fruitful field of research for specialists in these and other areas of applied mathematics.

References

[1] I. Aleksander and H. Morton, *An Introduction to Neural Computing* (Chapman & Hall, London, 1990).
[2] S.-i. Amari, Mathematical foundations of neurocomputing, *Proc. IEEE* **78** (9) (1990) 1143–1463.
[3] A. Ben-Israel and T.N.E. Greville, *Generalised Inverses, Theory and Applications* (Wiley, New York, 1974).
[4] R.W. Brause, Performance and storage requirements for topology-conserving maps for robot manipulator control, Report 5/89, Fachbereich Inform., Univ. Frankfurt, 1989.
[5] S.W. Ellacott, An analysis of the delta rule, in: *Proc. Internat. Neural Net Conf.*, Paris (Kluwer, Dordrecht, 1990) 956–959.

[6] S.W. Ellacott, The numerical analysis approach, in: J.G. Taylor, Ed., *Mathematical Approaches to Neural Networks*, North-Holland Math. Library **51** (North-Holland, Amsterdam, 1993) 103–138.
[7] E. Isaacson and H.B. Keller, *Analysis of Numerical Methods* (Wiley, New York, 1966).
[8] D. Jacobs, Ed., *The State of the Art in Numerical Analysis* (Academic Press, New York, 1977).
[9] A.J. Jones, Neural computing applications to prediction and control, Dept. Comput., Imperial College, London, 1992.
[10] J.C. Mason and P.C. Parks, Selection of neural network structures — some approximation theory guidelines, in: K. Warwick, G.W. Irwin and K.J. Hunt, Eds., *Neural Networks for Control and Systems*, IEE Control Engrg. Ser. **46** (Peter Peregrinus, London, 1992) 151–180.
[11] H.N. Mhaskar, Approximation properties of a multilayered feedforward artificial neural network, Dept. Math., California State Univ., Los Angeles, CA, 1991.
[12] E. Oja, *Subspace Methods of Pattern Recognition* (Research Studies Press, Letchworth, 1983).
[13] D.E. Rumelhart and J.L. McClelland, *Parallel and Distributed Processing: Explorations in the Microstructure of Cognition, Vols. 1 and 2* (MIT Press, Cambridge, MA, 1986).
[14] P.K. Simpson, *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations* (Pergamon, New York, 1990).
[15] G. Venkataraman and G. Athithan, Spin glass, the travelling salesman problem, neural networks and all that, *Prãmana J. Phys.* **36** (1) (1991) 1–77.
[16] Z. Wang, M.T. Tham and A.J. Morris, Multilayer feedforward neural networks: a canonical form approximation of nonlinearity, Dept. Chemical Process Engrg., Univ. Newcastle upon Tyne, 1992.
[17] P.D. Wasserman, *Neural Computing: Theory and Practice* (Van Nostrand Reinhold, New York, 1989).
[18] X. Yuan, W.A. Light and E.W. Cheney, Constructive methods of approximation by ridge functions and radial functions, 1992.