Techniques in Neural Learning

Pascal Koiran
DIMACS, Rutgers University[1]
e-mail: koiran@dimacs.rutgers.edu

John Shawe-Taylor
Department of Computer Science,
Royal Holloway, University of London,[2]
e-mail: john@dcs.rhbnc.ac.uk

NeuroCOLT Technical Report Series

NC-TR-95-036

April, 1995[3]

Produced as part of the ESPRIT Working Group
in Neural and Computational Learning,
NeuroCOLT 8556

NeuroCOLT Coordinating Partner

Royal Holloway
University of London

Department of Computer Science
Egham, Surrey TW20 0EX, England

For more information contact John Shawe-Taylor at the above address
or email neurocolt@dcs.rhbnc.ac.uk

[1]Piscataway, NJ 08855, USA
[2]Egham, Surrey, TW20 0EX, UK
[3]Received March 27, 1995

Abstract

This paper takes ideas developed in a theoretical framework by Maass [8] and adapts them for a practical learning algorithm for feedforward sigmoid neural networks. A number of different techniques are presented which are based loosely around the common theme of taking advantage of the linearity of the net input to a neuron, or in other words the fact that there is only a single non-linearity per neuron. Some experimental results are included, though many of the ideas are as yet untested. The paper can therefore be viewed as a tool box offering a selection of possible techniques for incorporation in practical, heuristic learning algorithms for multi-layer perceptrons.

1 Introduction

One of the major constraints on the application of Neural Network technology has been the difficulty in training large networks to perform useful tasks. This difficulty has become manifest both theoretically and practically, in that the loading problem has been shown to be NP-complete for very simple architectures [3] while in practical experiments even modest networks have been found to need inordinate amounts of computing power for successful training.

Recently, Maass undertook a theoretical investigation of the complexity of training neural networks of fixed size and succeeded in defining a class for which they are probably approximately learnable [8] (in Valiant's model for PAC-learning [10]). The key ingredients introduced by Maass are fixing the number of input lines to be a constant, and replacing sigmoid activation functions by piecewise polynomial ones. The algorithm makes critical use of this fact together with the fact that the net input to a neuron is a linear function. Hence, in one piece of the activation function, the overall functionality of the neuron will be polynomial. This allows the application of standard non-local techniques for adapting the weights.

The paper will include four different techniques more or less loosely centred around Maass's ideas together with a practical algorithm of Biegler-König and Bärmann, which similarly makes use of the restricted nature of the non-linearity of the neuron's functionality and an algorithm of Koiran, which though not apparently practical can be adapted to give a good heuristic approach to incremental learning of a network structure.

We summarise the techniques as follows.

1. The approach of regarding the activation function as a piecewise linear function suggests a way in which the linear programming algorithm applied to training feedforward networks [9] can be adapted to allow greater use of the local gradient information in each weight update. The key ingredient is to restrict the updates to ensure that the change in activation values at the hidden nodes is not too large. This in turn means that the gradient approximation will remain good. Note that the individual weight updates can be allowed to be large in contrast to standard backpropagation.

2. The second approach adheres more faithfully to Maass, in that a piecewise linear activation function is used. Once the initial weights have been chosen and the training examples given the boundaries of the non-linearities effectively partition weight space into a number of regions over which the error function is linear. Hence, within each region a linear programme can be solved to give the optimal choice of weights. This will be on the boundary of the region. The analysis in this paper then presents a technique for identifying the neighbouring hyperplane into which the gradient search should continue.

3. The third approach is similar to the second except that a quadratic error function is used. This results in the problem which must be solved in a region being a convex quadratic programme. Since, there are a polynomial time algorithms for such programmes, we can again find the optimal choice of weights in the given region efficiently. In this case we may find the optimum in the interior of the region. A similar technique to that described above can be used to find the next region if the optimal point is on the boundary.

4. The final technique is a heuristic version of Koiran's algorithm for incrementally adding sigmoid units to a single hidden layer network [7]. Each stage of the algorithm adds a new unit, while the output is given as a linear combination of the outputs of the hidden units. The expensive stage in the algorithm is that of finding the optimal choice for the parameters of the new hidden unit (which in Koiran's case is taken to be piecewise linear). The technique used is to enumerate all possible dichotomies generated by the breaks in linearity of the activation function. The number of these dichotomies is bounded by $O(n^d)$, where n is the number of examples and d is the input dimension. Hence, for even moderate dimensionality the algorithm is no longer practical. The heuristic method proposed here reverts to using a sigmoid functionality and a gradient descent technique to choose the weights of the additional neuron. However, by adapting the techniques of Biegler-König and Bärmann, the algorithm only has to contend with two non-linearities at each stage. This makes it possible to apply very fast gradient techniques to find a solution. It also means that there is less likelihood of local minima causing problems.

After a brief definitions section, the main body of the paper comprises two sections. The first presents the first three of the techniques as non-incremental algorithms, that is algorithms that do not adapt the network architecture. The final techniques is described in the later section on incremental learning. We finish with some conclusions and further thoughts.

2 Definitions

A feedforward network is a directed acyclic graph, in which vertices with positive in-degree represent computational nodes and those with in-degree zero represent input connections. An edge (j,i) from node j to node i represents a connection with corresponding weight w_{ij}. The function computed at a computational node i takes the weighted sum of the outputs o_j of the nodes j connected to node i and passes the result through an activation function $f(x)$:

$$\begin{aligned} o_i &= f(u_i) \\ \text{where} \quad u_i &= \sum_j w_{ij} o_j. \end{aligned}$$

In section 3.1, we will restrict consideration to the sigmoid activation function $f(x) = 1/(1+e^{-x})$. Piecewise-linewar activation functions will be used in sections 3.2 and 3.3.

The networks we will consider will be restricted to have a single output node and we denote the output of this node for input $\mathbf{i}$ and weight values $w = (w_{ij})$ by $F_{\mathcal{N}}(w, \mathbf{i})$, where $\mathcal{N}$ denotes the architecture of the network considered.

Following the approach presented by Werbos [11], we can view the backpropagation algorithm as computing the derivative of the error with respect to different parameters of the forward pass. Hence, the quantity that is computed during the backpropagation algorithm at node i, normally referred to as δ_i, is the partial derivative

$$\delta_i = \frac{\partial E}{\partial u_i},$$

where E is the output error on one or more presentations. The derivative that is actually required for updating the weights is that with respect to an individual weight

$$\frac{\partial E}{\partial w_{ij}} = \delta_i o_j.$$

3 Non-incremental learning techniques

This section deals with algorithms that attempt to optimize the weights of all units simultaneously on a fixed architecture (standard backpropagation falls in this category).

3.1 Restricting activation movements

We begin by giving some background concerning a related algorithm which will to a certain extent form the basis of the new approach.

3.1.1 The Linear Programming Algorithm

As mentioned by Shawe-Taylor and Cohen [9], a problem encountered by the back-propagation algorithm when solving discrete problems for which exact solutions lie "at infinity" in weight space is that as the weights increase one error can in fact tend to one rather than zero. This can be thought of as a local minimum at infinity. To overcome this problem they suggest that individual errors should not be allowed to increase. If the weight changes are given by δw, they require that

$$(F_{\mathcal{N}}(w+\delta w, \mathbf{i}^p) - t^p)^2 \leq (F_{\mathcal{N}}(w, \mathbf{i}^p) - t^p)^2,$$

where the training patterns are the pairs $(\mathbf{i}^p, t^p), p \in P$. Taking a first order approximation to this inequality:

$$\sum_{(k,l)} \frac{\partial E^p}{\partial w_{lk}} \delta w_{lk} \leq 0$$

for all $p \in P$, where δw_{lk} is the small change made to the weight on the connection (k,l) and E^p is the square error on sample p. The weight changes are then further restricted in order to minimise second order effects:

$$|\delta w_{lk}| \leq \epsilon_{lk},$$

where ϵ_{lk} are small non-negative quantities. Their learning requirement is to maximise the overal reduction in error subject to the above constraints. These requirements result in a linear programme which must be solved to perform one iteration of the algorithm, though note that the iteration is a batch iteration involving all (or at least a number of) the training samples. We should emphasise that the solution of the linear programme gives us the δw_{lk} which are small local changes to the weights satisfying the requirement that none of the individual errors should increase and, subject to that constraint, maximising the overall reduction in the error. As such they cannot be more accurate than the local information from which they are calculated.

3.1.2 The new algorithm

The linear programming algorithm essentially restricts the changes in the activations allowed at the output node for each input sample considered. As suggested by the Maass algorithm, if we view the activation functions as piecewise linear, the tangent can be seen as a good approximation to the sigmoid function in the region of that activation value. This suggests that by restricting the activations of individual computational nodes to remain close to their previous values for each of the sample inputs, the tangent approximation assumed in following gradient descent may remain valid even for large weight updates. We will show in the next section that this is indeed the case.

The resulting algorithm makes the following restrictions on the weight updates δw_{ij}.

$$\begin{aligned} \delta o_i^p &= \gamma_i \sum_{(j,i)} \delta w_{ij} o_j^p \leq \epsilon, & (1)\\ \text{and} \quad \delta o_i^p &\geq -\epsilon, & (2)\\ \text{where} \quad \gamma_i &= \sqrt{(o_i^p + \epsilon/4)(1 - o_i^p + \epsilon/4)} \end{aligned}$$

for all computational nodes i and samples $p \in P$ and some fixed small $\epsilon > 0$. Subject to these constraints the overall error should be reduced, possibly with an additional weight decay term to ensure that large weights receive some penalty and are therefore discouraged. The resulting set of equations, again defines a linear programme of approximately the same size as that considered by Shawe-Taylor and Cohen [9], since they obtained one constraint for each sample plus two for each weight, while in the above formulation there are two constraints for each sample, computational node pairing. One of the constraints on the network output node should in fact be dropped, since it restricts the error from falling too far, which is clearly undesirable. This detail is ignored in the subsequent analysis.

Each iteration of the algorithm involves performing backpropagation from the network output and then solving of the linear programme generated from the derivatives, and finally updating the weights with the solution δw_{lk} of the linear programme.

In the next section we will analyse the effect of the constraints and show that despite no restriction being placed on the size of the vector δw_{lk}, the movement remains in an area of weight space where the tangent approximation remains valid.

3.1.3 Analysis of the algorithm

We will estimate the difference between using a gradient estimation of the error at the point $w + \delta w$ and the actual error at this new point in weight space. Our aim is to show that this is bounded independently of the size of the vector δw, depending only on the size of the Hessian of the error with respect to the computational node outputs and the parameter ϵ. Our main result is the following.

Theorem 3.1 *If the weight update δw is constrained by equations 1 and 2, the difference between the estimate of the error and the actual error is bounded by*

$$\frac{1}{8}\epsilon^2 \sum_p \left(\sum_{i,i'} \left| \frac{\partial^2 E^p}{\partial o_i \partial o_{i'}}(o') \right| + 4 \sum_i \left| \frac{\partial E^p}{\partial o_i}(o') \right| \right),$$

where o' is a vector of outputs satisfying $|o_i - o'_i| < \epsilon/4$.

Proof. We use the Taylor expansion of $g(1)$ about 0, where $g(t) = E(w + t\delta w)$ for $t \in [0,1]$,

$$E(w+\delta w) = E(w) + gradE(w).\delta w + 0.5\delta w^T . \frac{\partial^2 E}{\partial w_{ij} \partial w_{i'j'}}(w).\delta w + \text{higher order terms}.$$

Equality holds because the function is analytic. Hence, the error of our estimate of $E(w + \delta w)$ can be bounded by

$$|E(w+\delta w) - E(w) - gradE(w).\delta w| \leq \left| 0.5\delta w^T . \frac{\partial^2 E}{\partial w_{ij} \partial w_{i'j'}}(w).\delta w \right.$$
$$\left. + \text{ higher order terms} \right|.$$

The result is obtained by bounding this error term independently of δw. To do this we first consider the formula for the gradient given in Section 2 and restrict ourselves to the case where the error is on just one sample input/output pairing.

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \delta_i o_j, \\ \text{where} \quad \delta_i &= \frac{\partial E}{\partial u_i}. \end{aligned}$$

Since the Hessian is symmetrical we can assume without loss of generality that $j \leq j'$. This implies that o_j will not depend on the weight $w_{i'j'}$. Hence, we can compute the second derivative as

$$\frac{\partial^2 E}{\partial w_{ij} \partial w_{i'j'}} = \frac{\partial^2 E}{\partial u_i \partial u_{i'}} o_j o_{j'},$$

using the chain rule. The same method can be used to convert the higher order expressions to include partial derivatives with respect to the u_i, since u_i is linearly dependent on w_{ij}. We now return to the expression

$$\mathcal{E} = 0.5 \delta w^T . \frac{\partial^2 E}{\partial w_{ij} \partial w_{i'j'}}(w).\delta w + \text{higher order terms.}$$

Written in terms of the individual components we have

$$\begin{aligned} \mathcal{E} &= 0.5 \sum_{i,j} \delta w_{ij} \sum_{i',j'} \delta w_{i'j'} \frac{\partial^2 E}{\partial w_{ij} \partial w_{i'j'}}(w) + \text{higher order terms} \\ &= 0.5 \sum_{i} \sum_{i'} \frac{\partial^2 E}{\partial u_i \partial u_{i'}} \sum_{j} \delta w_{ij} o_j \sum_{j'} \delta w_{i'j'} o_{j'} + \text{higher order terms.} \end{aligned}$$

Letting

$$\delta u_i = \sum_j \delta w_{ij} o_j,$$

we can write

$$\mathcal{E} = 0.5 \sum_{i} \sum_{i'} \frac{\partial^2 E}{\partial u_i \partial u_{i'}} \delta u_i \delta u_{i'} + \text{higher order terms.}$$

The higher order terms are similarly dependent on this quantity and together they form the second and higher order terms in the Taylor expansion of $E(u+\delta u)$ about $E(u)$, where $u = (u_i)$ is a vector of net inputs of the individual nodes and $\delta u = (\delta u_i)$ is the vector of adaptions introduced at node i. It should be understood that the effect of the δu_i's accumulate to give an overall change in the error. However, again because the function is analytic the series is equal to the error and so we can now use the standard expression for the error term of a curtailed Taylor series to write

$$\mathcal{E} = 0.5 \sum_{i} \sum_{i'} \frac{\partial^2 E}{\partial u_i \partial u_{i'}}(u + \alpha' \delta u) \delta u_i \delta u_{i'},$$

for some $\alpha' \in (0,1)$. Using the chain rule we can write

$$\frac{\partial^2 E}{\partial u_i \partial u_{i'}}(u + \alpha'\delta u) = \frac{\partial^2 E}{\partial o_i \partial o_{i'}}(o')o'_i(1 - o'_i)o'_{i'}(1 - o'_{i'}),$$

for $i \neq i'$, while

$$\frac{\partial^2 E}{\partial u_i \partial u_i}(u + \alpha'\delta u) = \frac{\partial^2 E}{\partial o_i \partial o_i}(o')(o'_i)^2(1 - o'_i)^2 + \frac{\partial E}{\partial o_i}(o')(1 - 2o'_i)o'_i(1 - o'_i),$$

where $o'_i = f(u_i + \alpha'\delta u_i)$. Since $|\delta u_i \sqrt{(o_i + \epsilon/4)(1 - o_i + \epsilon/4)}| \leq \epsilon$, it can be shown that $|\delta u_i \sqrt{o'_i(1 - o'_i)}| \leq \epsilon$, and hence

$$|\mathcal{E}| \leq \frac{1}{8}\epsilon^2 \left(\sum_i \sum_{i'} \left| \frac{\partial^2 E}{\partial o_i \partial o_{i'}}(o') \right| + 4 \sum_i \left| \frac{\partial E}{\partial o_i}(o') \right| \right),$$

giving the result in the case where the error is measured on a single input/output pairing. Finally, consider the case where

$$E(w) = \sum_{p \in P} E^p(w).$$

In this case we can write the difference between the actual error and the estimate of the error as the sum over the contributions from the individual input/output pairings, giving the overall result. $\square$

We will discuss the implications of this theorem in the conclusions section.

3.2 Using a piecewise-linear error function

In this paper a continuous function $f : \mathcal{R}^m \to \mathcal{R}$ is said to be piecewise-linear if the input space can be partitioned into a finite set of closed convex polyhedra such that f is affine on each polyhedron (overlapping between the boundaries of the polyhedra is allowed).

If the error function $E(u)$ is a piecewise-linear function of the network weights u, then one can directly apply linear programming techniques without making any kind of approximation. Indeed, if P is a region of the weight space of the form $Au \leq b$ where the current weight vector u^0 is located, and if E is affine on P, then by definition E can be optimized on P by solving the linear "program"

$$\min_{Au \leq b} E(u),$$

e.g., by the simplex method. The optimum is reached on a vertex u^1 of P. One must then decide if u^1 is a local minimum, by optimizing E on the regions of linearity of E that contain u^1. One can for instance choose the first region where an improvement is obtained. This generates a new point u^2. Then this procedure is repeated with u^2 as new starting point. If no improvement can be obtained, then the algorithm stops and outputs the local minimum u^1. The advantages of this method over conventional backpropagation are that:

- ◇ convergence to a local minimum is guaranteed.
- ◇ there is no need for tuning the step size or other parameters.
- ◇ the algorithm may be able to take "big" steps if the current region is "big". This has the potential to reduce the number of iterations needed to reach a local minimum.

Another advantage of the linear programming approach is that it is straightforward to train networks that never underestimate the target function on the training data. This condition can be enforced by simply adding the corresponding inequalities to the linear system. Depending on the application, it may be desirable to overestimate the target function in this way. When this is not the case, one can train two networks F^- and F^+. The latter network will overestimate the target values, and the former will underestimate. On unseen examples $(x, F^*(x))$ of the target function F^*, $[F^-(x), F^+(x)]$ will be a confidence interval for the true value $F^*(x)$. The degree of confidence can be estimated by a Vapnik-Chervonenkis dimension argument.

We explain in section 3.2.2 how the set of regions containing u^1 can be explored efficiently. We give a few examples of neural networks with a piecewise-linear error function in the next section.

3.2.1 Some networks with a piecewise-linear error function

The error of a function $F : \mathcal{R}^d \to \mathcal{R}$ on a finite set of examples $\{(x_1, y_1), \ldots, (x_s, y_s)\} \subset \mathcal{R}^d \times \mathcal{R}$ will be measured by

$$E(F) = \sum_{i=1}^{s} |F(x_i) - y_i|. \tag{3}$$

The function F is assumed to be a one-hidden-layer neural network of the form

$$F(x) = \sum_{i=1}^{n} a_i \phi(\langle w_i, x\rangle - \theta_i) \tag{4}$$

where $\langle ., . \rangle$ denotes the inner product in $\mathcal{R}^d$. It is sometimes convenient to add an output threshold. In this case, the function computed by the network is $F'(x) = F(x) + \theta$, where F is of the form (4). In sections 3.2 and 3.3, we will consider only piecewise-linear output functions ϕ. The linear bounded sigmoid

$$\pi(x) = \begin{cases} 1 \text{ if } x \geq 1; \\ x \text{ if } -1 \leq x \leq 1; \\ -1 \text{ if } x \leq -1. \end{cases}$$

is of particular interest. The piecewise-linearity of ϕ implies that E is a piecewise-linear of the data x_i, y_i. However, the property in which we are really interested is piecewise-linearity in the network parameters a_i, w_i and θ_i. In the case

$\phi = \pi$, this can be achieved thanks to a parameter transformation which was introduced in [8]. For any $a \geq 0$, let π_a be defined by:

$$\pi_a(x) = \begin{cases} -a \text{ for } x \leq -a; \\ x \text{ for } -a \leq x \leq a; \\ a \text{ for } x \geq a. \end{cases} \tag{5}$$

If $a > 0$, $\pi_a(x) = a\pi(x/a)$. The contribution of hidden unit i to F is $a_i\pi(\langle w_i, x\rangle - \theta_i)$. Since π is odd, one can assume without loss of generality that $a_i \geq 0$. This contribution is equal to $\pi_{a_i}(\langle w'_i, x\rangle - \theta'_i)$, where $w'_i = a_i w_i$ and $\theta'_i = a_i\theta_i$. The advantage of this parametrization is that F is now a piecewise-linear function of the new parameters a_i, w'_i, θ'_i.

Note that no parameter transformation is needed if the output function satisfies $\phi(ax) = a\phi(x)$ for every $a > 0$. In this case one can assume without loss of generality that the output weights are in $\{-1, 1\}$. An example of such a function is $\phi(x) = x + |x|$.

Another drastic way to deal with the non-piecewise-linearity of (4) is to fix each of the output weights a_i to the same a priori value $a_i = B/n$ (if ϕ is not odd, one may also have to allow the opposite value $a_i = -B/n$). The total weight $B = \sum_{i=1}^n |a_i|$ should be related to the smoothness of the function to be learnt (this can be made precise). Of course, if learning is not succesful for a given value of B, one can try again with a larger value (set for instance $B := 2B$). This may seem at first sight to be overly restrictive, but in fact these networks still have good approximation properties. See for instance [2, 4, 7] and section 3.3 for the L^2 case; see [1, 4, 5] for the L^∞ case (the L^∞ results in these papers are formulated only for threshold functions, but it can be shown that they apply to other output functions, and to π in particular).

Assume for instance that the linear bounded sigmoid π is used, or rather π_a after the parameter transformation (5). The regions of linearity of F are defined by the $2sn$ hyperplanes of $\mathcal{R}^{n(d+2)}$ of equations

$$\langle w_j, x_i\rangle - \theta_j = \epsilon_{ij} a_j \ (i = 1, \ldots, s;\ j = 1, \ldots, n;\ \epsilon_{ij} = -1, 1). \tag{6}$$

In order to obtain the regions of linearity of E, these regions must be refined to take into account the absolute values in (3). Indeed, for each example x_i, $F(x_i) - y_i$ might be either positive or negative. Each region of E could therefore be divided into 2^s subregions. This can be avoided by the following trick: introduce for each i two new variables Δ_i and Δ_i^+. Minimizing (3) is equivalent to minimizing

$$E' = \sum_{i=1}^{s}(2\Delta_i^+ - \Delta_i) \tag{7}$$

subject to

$$\Delta_i = F(x_i) - y_i;\ \Delta_i^+ \geq 0;\ \Delta_i^+ \geq \Delta_i.$$

Indeed, the last two constraints imply that Δ_i^+ is not smaller than the positive part of Δ_i, and $E = E'$ whenever Δ_i^+ is equal to the positive part of Δ_i.

3.2.2 Travelling through weight space

Let m be the number of network parameters and let $u^0 \in \mathcal{R}^m$ be the current weight vector. Assume that u^0 is a vertex of a convex polyhedron P where the error function E is linear. A basic task is to determine whether u is a local minimum of E. This point is at the intersection of p hyperplanes $l_1(x) = 0, \ldots, l_p(x) = 0$ which bound the regions of linearity of E ($p = m$ in general). The number of regions containing u^0 can therefore be exponential[1] in m. This will make an exhaustive search impossible even for moderate values of m. It may thus be necessary to resort to heuristics that explore only part of the neighbouring regions (it would be interesting to establish a NP-hardness, or rather a co-NP-hardness result for this local minimum problem). There is one important case when exhaustive search can provably be avoided. This occurs in non-degenerate situations, i.e., when a vertex of P is the intersection of exactly m hyperplanes.

Theorem 3.2 *Let $H_1, \ldots, H_m$ be m hyperplanes of $\mathcal{R}^m$ such that*

$$\bigcap_{i=1}^{m} H_i = \{x_0\}$$

for some $x_0 \in \mathcal{R}^m$. Let f be a continuous function which is affine in every region defined by $H_1, \ldots, H_m$. If x_0 is not a minimum of f, then there is a point x_1 in the intersection of m-1 of the H_i's such that $f(x_1) < f(x_0)$.

Note that by linearity x_0 is a local minimum if and only if it is a global minimum.

Proof of Theorem 3.2. Let us assume for convenience that $x_0 = 0$. The equation of each H_i is of the form $l_i(x) = 0$, where l_i is a linear form. Let x'_0 be a point such that $f(x'_0) < f(0)$, and let $\mathcal{R}$ be a region containing x'_0. Since $f - f(0)$ is linear on $\mathcal{R}$ and the forms $l_1, \ldots, l_m$ are linearly independent, there are coefficients $\alpha_1, \ldots, \alpha_m$ such that

$$f(x) = \sum_{i=1}^{m} \alpha_i l_i(x) + f(0)$$

on $\mathcal{R}$. Since $f(x'_0) < f(0)$, there exists j such that $\alpha_j l_j(x'_0) < 0$. Let x_1 be any point such that $l_i(x_1) = 0$ for $i \neq j$ and $\alpha_j l_j(x_1) < 0$. Then $x_1 \in \mathcal{R}$ since for every $i \in \{1, \ldots, m\}$, $l_i(x'_0)$ and $l_i(x_1)$ have the same sign. Hence $f(x_1) - f(0) = \alpha_j l_j(x_1) < 0$. $\square$

Let L_i^+ be the half-line of equation

$$l_i \geq 0;\ l_j = 0 \text{ for } j \neq i$$

and let L_i^- be the half-line of equation

$$l_i \leq 0;\ l_j = 0 \text{ for } j \neq i,$$

[1] To convince yourself that this can be the case, note that the m hyperplanes $x_1 = 0, x_2 = 0, \ldots x_m = 0$ determime 2^m regions which intersect at the origin.

where as in the above proof, $l_i = 0$ is the equation of H_i. According to Theorem 3.2, it is sufficient to check the $2n$ half-lines $L_1^+, \ldots, L_n^+, L_1^-, \ldots, L_n^-$ in order to determine whether x_0 is a local minimum. It is perhaps interesting to note that these $2n$ half-lines can be replaced by $n+1$ regions.

Corollary 1 *The point x_0 is a minimum of f if and only if it is a minimum of f restricted to the domain*

$$\mathcal{R} = \mathcal{R}^+ \cup \bigcup_{i=1}^{n} \mathcal{R}_i^-,$$

where $\mathcal{R}$ is the region

$$l_j \geq 0 \textit{ for } j = 1, \ldots, n$$

and $\mathcal{R}_i^-$ is the region

$$l_i \leq 0;\ l_j \geq 0 \textit{ for } j \neq i.$$

Proof. It is easily verified that the $2n$ half-lines $L_1^+, \ldots, L_n^+, L_1^-, \ldots, L_n^-$ are included in these $n+1$ regions. □

In a degenerate situation, one reasonable heuristic might be to consider only those regions that share a face of maximum dimension (i.e., dimension $m-1$) with the current region. Note that by Corollary 1, this heuristic exactly determines whether the current point x_0 is a local minimum in a non-degenerate situation ($\mathcal{R}$ plays the role of the current region).

When the activation functions π_a are used, the weight vector

$$w_j = 0;\ \theta_j = 0;\ a_j = 1\ (j = 1, \ldots, n)$$

is a reasonable starting point for the local search procedure studied in this section. This choice ensures that each pattern x_i is in the linear part of the activation function of each hidden unit.

3.3 The case of a quadratic error function

In this section, we examine the consequences of using the quadratic error

$$E(F) = \sum_{i=1}^{s} [F(x_i) - y_i]^2 \tag{8}$$

instead of (3). Of course one cannot expect (8) to be piecewise-linear. It is thus impossible to use linear programming techniques. However, if F is piecewise-linear, then E is a piecewise convex quadratic function. Since there are efficient algorithms for convex quadratic programming under linear constraints, much of the rest of the paper applies in this case.

One of the advantages of using (8) is that contrary to a linear function, a convex quadratic function can reach its (strict) minimum in the interior of its domain. When this happens, the minimum is automatically a local minimum of (8) and there is no need to check neighbouring regions. Even if the minimum is reached on a face of dimension $m-k$, there are only 2^k neighbouring regions in a generic situation (k is always equal to m in the linear case).

Another advantage is that there is no need to introduce the additional variable Δ_i^+ as in (7), since (8) is convex quadratic on each region of F.

The following result is a generalization of Theorem 3.2 to piecewise smooth and convex functions. It applies in particular to the quadratic error function.

Theorem 3.3 *Let $H_1, \ldots, H_m$ be m hyperplanes of $\mathcal{R}^m$ such that*

$$\bigcap_{i=1}^{m} H_i = \{x_0\}$$

for some $x_0 \in \mathcal{R}^m$. Let f be a continuous function which is C^1 and convex on every region defined by $H_1, \ldots, H_n$. If x_0 is not a minimum of f, then there is a point x_1 in the intersection of m-1 of the H_i's such that $f(x_1) < f(x_0)$.

Proof. Let us assume for convenience that $x_0 = 0$. The equation of each H_i is of the form $l_i(x) = 0$, where l_i is a linear form. Let x'_0 be a point such that $f(x'_0) < f(0)$, and assume for convenience that $l_i(x'_0) \geq 0$ for every $i = 1, \ldots, s$. Let us perform the change of variable $y_i = l_i(x)$: there is a convex and C^1 function $g : [0, +\infty[^m \rightarrow \mathcal{R}$ such that $g(y) = f(x)$. Let $y'_{0i} = l_i(x'_0)$ and let $h : [0,1] \rightarrow \mathcal{R}$ be defined by: $h(t) = g(ty'_0)$. Since h is convex and $h(1) < h(0)$, $h'(0) < 0$. Since g is C^1, $h'(0) = \sum_{i=1}^{m} y'_{0i} \frac{\partial g}{\partial y_i}(0)$. This implies that $\frac{\partial g}{\partial y_j}(0) < 0$ for some $j \in \{1, \ldots, s\}$. Hence $g(y'_1) < g(0)$ if $y'_{1j} > 0$ is small enough and $y'_{1i} = 0$ for $i \neq j$. The desired point x_1 such that $x_{1i} = l_i(y_1)$ clearly satisfies $f(x_1) < f(x_0)$ and is in $\bigcap_{i \neq j} H_i$. □

Corollary 1 also clearly applies in this case.

Note that Theorem 3.3 can fail if the smoothness condition is removed. Consider for instance the function $f : \mathcal{R}^2 \rightarrow \mathcal{R}$ such that $f(x,y) = 0$ if $x \leq 0$ or $y \leq 0$, $f(x,y) = -x$ for $y \geq x \geq 0$ and $f(x,y) = -y$ for $x \geq y \geq 0$.

The greatest advantage of the quadratic loss function is perhaps that it is possible to use iterative learning algorithms. These algorithms are analyzed.

3.4 Degeneracy analysis

The significance of theorems 3.2 and 3.3 stems from the fact that the intersection of more than m "random" hyperplanes in $\mathcal{R}^m$ is empty with probability one. However the hyperplanes defining the regions of E are not random since they come from the very specific error functions (3) and (8). One can therefore ask whether these two theorems apply when the data (x_i, y_i) are chosen at random. Only the quadratic error function will be considered here, but a similar analysis can be performed for (3).

In the quadratic case, the y_i's do not play any role. Recall that if the activation functions π_a are used, the hyperplanes of interest are defined by (6). Let us consider the weight space of hidden unit j. For any choice of the x_i's, the line of equation

$$w_j = 0;\ \theta_j = t;\ a_j = -t$$

is in the intersection of the s hyperplanes

$$\langle w_j, x_i \rangle - \theta_j = a_j\ (i = 1, \ldots, s)$$

and the line of equation

$$w_j = 0;\ \theta_j = t;\ a_j = t\ (i = 1, \ldots, s)$$

is in the intersection of the s hyperplanes

$$\langle w_j, x_i \rangle - \theta_j = -a_j.$$

It can be shown that except for a set of x_i's of Lebesgue measure 0 in $\mathcal{R}^{ds}$, these are the only degeneracies that can occur.

4 Incremental learning

Incremental learning algorithms attempt to optimize the weights of only one hidden unit at a time. This has the potential to reduce considerably the computational cost of the algorithm. Jones [6] proposed an iterative learning scheme with a guaranteed error bound. His result was improved by Barron [2], who showed that the weights of each unit need only be optimized approximately. Then Darken et al. [4] showed that the output weights of the network can be kept constant, and Koiran [7] showed that the same approximation rate can be achieved even if the target function is not exactly realizable. His result was as follows:

Theorem 4.1 *Let G be a subset of a Hilbert space H, with $||g|| \leq B$ for each $g \in G$.*
Let $m = \inf_{F \in co<(G)} ||F - f||$, where $f \in H$ and $co(G)$ is the convex hull of G. Then for every every $c > \sqrt{B^2 + m^2}$,

$$||f_n - f|| \leq m + \frac{c}{\sqrt{n}}$$

where f_n, $n = 1, 2, \ldots$ is any sequence chosen to satisfy

$$||f_1 - f||^2 \leq \inf_{g \in G} ||g - f||^2 + \epsilon_1$$

and, for $n \geq 2$,

$$||f_n - f||^2 \leq \inf_{g \in G} ||\alpha_n f_{n-1} + \overline{\alpha}_n g - f||^2 + \epsilon_n \tag{9}$$

where $\alpha_n = (n-1)/n$, $\overline{\alpha}_n = 1 - \alpha_n = 1/n$, and $\epsilon_n \leq (c^2 - B^2 - m^2)/n^2$.

In applications to neural learning, $||f_n - f||^2$ will represent the quadratic error of the network f_n on the training set. It is clear that one can (heuristically) use the techniques of section 3.3 to solve problem (9), since it is equivalent to the training problem for a network with a single hidden unit.

We consider the incremental approach suggested by Theorem 4.1. The space G will be the set of functions realisable as single sigmoid neurons with output multiplied by a constant of size at most B,

$$G = \{A\sigma(\langle w_i, x\rangle) | A \leq B\},$$

where $\sigma(x)$ is the sigmoid function and we assume that the threshold has been incorporated into the weight vector by augmenting the input with a coordinate whose value is always 1. Now f_n will be a convex combination of n functions from G and in particular

$$f_n = (g + (n-1)f_{n-1})/n,$$

for some $g \in G$. The task of the incremental stage of the algorithm will be to find g_0 given f_{n-1} such that

$$\|\alpha f_{n-1} + \bar{\alpha} g_0 - f\|^2 \leq \inf_{g \in G} \|\alpha f_{n-1} + \bar{\alpha} g - f\|^2 + \eta_n,$$

where $\alpha + \bar{\alpha} = 1$, $\bar{\alpha} = 1/n$, f is the target and $\eta_n = 1/n^2$. As in Koiran [7], we consider the space defined by the inputs x_i weighted by their frequency if they are repeated. Hence, the norm is determined exclusively by the values of the functions at these points. Note that we are only asking that the function we choose be at least as good as the best we could choose at this stage with a margin of $1/n^2$ error.

We will not go further into the analysis of what such an algorithm would deliver in terms of generalisation, but concentrate on defining an efficient heuristic for finding g_0. The function g_0 is determined by a weight vector w and output multiplier a. It is these values that the algorithm must compute.

Note first that for each input/output pairing (x_i, y_i), the error contributed by that example for a particular choice of w and a is

$$(\alpha f_{n-1}(x_i) + \bar{\alpha} a\sigma(\langle w, x_i\rangle) - y_i)^2 = \frac{1}{n^2}(a\sigma(\langle w, x_i\rangle) - z_i)^2,$$

for $z_i = ny_i - (n-1)f_{n-1}(x_i)$, which is independent of a and w. Hence, the problem is equivalent to minimising

$$L(w,a) = \sum_i (a\sigma(\langle w, x_i\rangle) - z_i)^2,$$

to within a margin of error equal to 1. Computing $\mathrm{grad}_w L(w,a)$, we have

$$\mathrm{grad}_w L(w,a) = \sum_i 2\sigma'(\langle w, x_i\rangle) x_i (a\sigma(\langle w, x_i\rangle) - z_i).$$

If we treat $v_i = \sigma(\langle w_0, x_i\rangle)$ and $v_i' = \sigma'(\langle w_0, x_i\rangle) = v_i(1 - v_i)$ as constants (w_0 is the current value of w) and let $u_i = \sigma^{-1}(z_i/a)$, we obtain the approximation

$$\text{grad}_w L(w, a) = \sum_i 2v_i' x_i a v_i''(\langle w, x_i\rangle - u_i),$$

where $v_i'' = (\sigma(\langle w_0, x_i\rangle) - z_i/a)/(\langle w_0, x_i\rangle - u_i)$. Solving for $\text{grad}_w L(w, a) = 0$, and approximating v_i'' by v_i' we obtain

$$\sum_i {v_i'}^2 x_i(\langle w, x_i\rangle) = \sum_i {v_i'}^2 x_i u_i$$

or a set of linear equations with coefficient matrix

$$B = \sum_i {v_i'}^2 x_i x_i^T,$$

and right hand side

$$c = \sum_i {v_i'}^2 x_i u_i.$$

Note that this is equivalent to backpropagating the required outputs through the sigmoid (to give u_i) and doing a least square approximation to this required net input value with the equation i weighted by $\sigma'(\langle w_0, x_i\rangle)^2$. After determining w, we solve for a treating the new value of w as a constant. We obtain

$$\text{grad}_a L(w, a) = 2\sum_i \sigma(\langle w, x_i\rangle)(a\sigma(\langle w, x_i\rangle) - z_i) = 0,$$

giving

$$a = \frac{\sum_i v_i z_i}{\sum_i v_i^2}.$$

5 Conclusions

The purpose of this paper has not been to present a particular finished algorithm, but rather a number of useful techniques all of which might be relevant to the synthesis of efficient adaptive techniques for neural networks. There are several themes underlying the different approaches. Perhaps the most important is the use of the linearity of the net input to a neuron. This allows for more global adaption of the parameters than is possible if we assume that the number of non-linearities is equal to the number of parameters, as for example in standard back-propagation.

Hence, for example we have shown that placing certain constraints on the weight updates ensures that error using the gradient estimate remains close to the actual error, provided the gradient and the Hessian of the error with respect to the outputs of individual neurons remain small compared with $1/\epsilon$. In this sense, we are able to be confident that the steps taken in the weight updates are justified. There is no reason to believe that the gradient or the Hessian should become large, indeed we conjecture that they both remain small. We would therefore expect that taking a value of $\epsilon = 0.1$ should keep the error bounds in the region of 0.01, while allowing very significant movements in weight space compared with those given by standard backpropagation.

In the second and third techniques once a significant region of linearity has been identified we can use global techniques such as linear programming to find an optimal solution in that region. The question then addressed is how to cross the boundary between regions efficiently.

The final technique adapts a polynomial time (for constant input dimension) algorithm for incremental learning to an efficient heuristic algorithm, in which the number of linearities in each search stage is just two. This results in very tractable optimisations, with a low probability of problems with local minima.

References

[1] A. Barron. Neural net approximation. In *Proc. 7th Yale Workshop on Adaptative and Learning Systems*, 1992.

[2] A. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.

[3] A. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. In *Proc. 1988 Workshop on Computational Learning Theory*, pages 9–18. San Mateo: Morgan Kaufmann, 1988.

[4] C. Darken, M. Donahue, L. Gurvits, and E. Sontag. Rate of approximation results motivated by robust neural network learning. In *Proc. 6th Annual ACM Conference on Computational Learning Theory*, 1993.

[5] L. Gurvits and P. Koiran. Approximation and learning of convex superpositions. To appear in *Proc. 2nd European Workshop on Computational Learning Theory*, 1995.

[6] L. K. Jones. A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training. *Annals of Statistics*, 20:608–613, March 1992.

[7] P. Koiran. Efficient learning of continuous neural networks. In *Proc. 7th ACM Conference on Computational Learning Theory*, pages 348–355, 1994.

[8] W. Maass. Bounds for the computational power and learning complexity of analog neural nets. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 335–344, 1993.

[9] J. S. Shawe-Taylor and D. A. Cohen. The linear programming algorithm for neural networks. *Neural Networks*, 3:575–582, 1990.

[10] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[11] P. J. Werbos. *The Roots of Backpropagation*. John Wiley and Son, 1993.