The Linear Programming Algorithm for Neural Networks

John S. Shawe-Taylor and Dave A. Cohen
Department of Computer Science
Royal Holloway and Bedford New College
Egham Surrey TW20 0EX UK

16th June, 1989

For reprint requests contact John Shawe-Taylor at the above address. Tel: (0784) 439021.

Running title: Linear Programming Algorithm

The Linear Programming Algorithm for Neural Networks

Abstract

A new learning algorithm for feed-forward neural networks based on linear programming is introduced. This alternative to back-propagation gives faster and more reliable learning on reasonably sized examples. Extensions of the method for efficient (approximate) implementations in large networks are considered.

Keywords: Neural networks, back-propagation, learning, feed-forward, linear programming algorithm, parallel distributed processing, network simulations.

1 Introduction

Back-propagation using the generalised delta rule was introduced as the solution to the credit-assignment problem which existed for threshold neurons [Ru-Hi-W86], thus opening up the possibility of learning in multi-layer perceptron networks. Back-propagation (BP) is an efficient method of calculating the partial derivatives of the error with respect to changes in the weights, and then using a gradient descent algorithm which changes the weights in the direction of fastest error reduction over all stimuli simultaneously. The algorithm is criticised by Minsky and Pappert [Mi-Pa88] since not only is there no guarantee that the algorithm will not get trapped in a local minimum, but also it is not clear that it represents a significant speed up over random weight assignments in realistically sized examples.

These criticisms are, perhaps, overstated. Though local minima do exist [He88], they seem in most cases not to be a problem. The PDP book [Mc-Ru86] does present many interesting examples where the BP algorithm has successfully learnt certain representations. There are, however, causes for concern when for one reasonably small example the method is reported to work in only 95% of the initial conditions tried.

Failures are usually reported for discrete problems, that is those whose input-output specification pairs are all 0,1 vectors. This means that their solutions can only ever be approximated by finite weight assignments. In these cases back-propagation can tend in the wrong "direction" in weight space, allowing at least one of the errors to grow rather than decrease.

Motivated by this observed problem, this paper derives a new learning algorithm by specifying that the local weight changes must not allow any of the errors to increase. The resulting algorithm is called the Linear Programming Algorithm, since at each iteration a linear programme must be solved. That it solves the incorrect direction problem mentioned above is borne out by examples. Use of a linear programme has been made for learning in spin-glass-like neural networks [Kr-Me87] in order to maximise basins of attraction, but the authors are unaware of its application to feed-forward Networks. The new algorithm is also a gradient descent algorithm, but is seen as more circumspect about the route down it takes. Any local minima will be as acute a problem for the new algorithm as for back-propagation. Indeed we are able to characterise when the new algorithm fails to

find a solution. This characterisation suggests that for non-correlated inputs the number of presentations can be as large as the number of edges divided by the number of output nodes. For inputs with significantly inter-correlated groups which are required to give the same output (e.g. all examples of one pattern in a classification problem) the number will typically be higher.

The paper has a short second section introducing notation for feedforward networks followed by the main section which develops the Linear Programming algorithm. In section 4 some experimental results obtained using the new algorithm are described. This is followed by a forward looking section suggesting ways in which the algorithm might be efficiently approximated and hence realistically implemented for large networks. The paper finishes with some general observations.

2 Definitions

At each time instant all PDP units in our model have an output value associated with them that is a function of the inputs to that unit. This output is computed by passing a weighted sum of the unit's inputs through the unit's activation function. Each unit has a distinguished input that always has value 1, called the threshold input. The following is a formal notation for these systems.

We will assume for simplicity that the activation function is the same function for all nodes of the network and we denote the function by f. Note that all of the following theory would still apply if each unit had a distinct activation function. So we have

$$f : \mathcal{R} \to \mathcal{I},$$

which is traditionally a monotonic function where $\mathcal{R}$ denotes the set of real numbers and $\mathcal{I}$ a finite interval on the real line. This interval is usually taken as $[0, 1]$ or $[-1, 1]$. A network $\mathcal{N} = (N, I, O, n_0, E)$ is specified by disjoint sets N and I of nodes, where I will be the input nodes, and a subset $O \subseteq N$ of output nodes and a node $n_0 \in I$, called the threshold node. The connectivity is given by a set $E \subseteq (N \cup I) \times N$ of connections, with $\{n_0\} \times N \subseteq E$, that is all non-input nodes are connected to from the threshold node.

With network $\mathcal{N}$ we associate a weight function on the set of connections:

$$w : E \to \mathcal{R}.$$

We say that the network $\mathcal{N}$ is in state w.

For notational convenience choose a numbering for all nodes and refer to each node by its number. For a connection $(i, j) \in E$ we denote $w(i, j)$ by w_{ji}. It is often convenient to form a square matrix W of weight values with rows and columns indexed by the set N of nodes. The entries in the matrix are given by

$$W_{ji} = \begin{cases} w_{ji}; & \text{if } (i,j) \in E, \\ 0; & \text{otherwise.} \end{cases}$$

Note that weight values associated with connections from input nodes do not occur in this matrix.

For the purposes of this paper we will assume throughout that if node i is connected to node j then $j > i$. Such a numbering can always be found provided the network is cycle free. This is the feedforward condition on the connectivity and though much of the analysis can be applied, the practical problems of stability without this condition are not addressed. This feedforward condition implies that the matrix W is lower triangular with zero diagonal.

At a given time the input values to the whole network are specified by a function $\mathbf{i}$ from the set of input nodes other than n_0 to $\mathcal{I}$:

$$\mathbf{i} : I \setminus \{n_0\} \to \mathcal{I}.$$

Each node also has an output value associated with it at each time instant. These values are given by the function:

$$o : N \cup I \to \mathcal{I}.$$

For notational convenience whenever a function takes values on nodes we often write the argument as a subscript. Having introduced the notation we can now formulate the well known equations governing the inputs, outputs and intermediate values in a network. The value of the weighted sum of the inputs to each node of N is given by a function u:

$$u : N \to \mathcal{R},$$

where

$$u_j = \sum_{(i,j)\in E} w_{ji} o_i, \quad \text{for } j \in N.$$

The outputs of individual nodes are then given by:

$$o_j = \begin{cases} \mathbf{i}_j; & \text{if } j \in I \setminus \{n_0\}, \\ 1; & \text{if } j = n_0, \\ f(u_j); & \text{if } j \in N, \end{cases}$$

and the outputs of the whole network are given by:

$$\mathbf{o}_j = o_j, \quad \text{for } j \in O.$$

Given the feedforward condition the network $\mathcal{N}$ always determines a function

$$F_{\mathcal{N}} : \mathcal{R}^{|E|} \times \mathcal{I}^{|I|-1} \to \mathcal{I}^{|O|},$$

given by

$$F_{\mathcal{N}}(w, \mathbf{i}) = \mathbf{o}.$$

Provided that the activation function f is differentiable this function will also be.

3 Linear Programming Algorithm

This section develops some of the theory of learning in feedforward networks necessary to derive the Linear Programming algorithm. We begin with a subsection on learning in networks before introducing the new ideas in the second subsection 'Derivation of the Linear Programming Algorithm'.

3.1 Learning in Networks

When specifying the behaviour required of a network we generally give a sequence of inputs with corresponding required outputs. These response pairs are indexed by a set P of presentations. Hence for $p \in P$, $\mathbf{i}^p$ is the input vector and $\mathbf{o}^p$ is the corresponding output vector. We call a set of presentations discrete if the input and output vectors all have values only 0 or 1.

For a given network $\mathcal{N}$ and presentation p, there is an error vector

$$\mathbf{e}^p = F_{\mathcal{N}}(w, \mathbf{i}^p) - \mathbf{o}^p.$$

By learning in a network we mean modifying the state (weight function) of the network. The central aim of a learning algorithm for network $\mathcal{N}$ and set P of presentations is to choose the weight function w so as to minimise in some sense the error vectors $\mathbf{e}^p$ for all presentations $p \in P$.

The back-propagation algorithm is a gradient descent method used to minimise the scalar error function

$$\sum_{p \in P} \|\mathbf{e}^p\|_2^2.$$

Rather than commit ourselves to this particular error function we prefer to evaluate the partial derivatives of the individual components of the error function. We can then combine them to recover the back-propagation algorithm if we wish, but at the same time have greater insight into how the different presentations are contributing to the weight changes.

Taking partial derivatives with respect to w_{lk} and letting

$$\Delta_{jl} = \frac{1}{o_k} \frac{\partial u_j}{\partial w_{lk}}, \tag{A}$$

we obtain that Δ_{jl} satisfies

$$\Delta_{jl} = \sum_{(i,j) \in E} w_{ji} f'(u_i) \Delta_{il} + \delta_{jl}, \quad \forall j,$$

and is independent of k. Hence if D is the square diagonal matrix with entries $D_{ii} = f'(u_i)$, for $i \in N$, and Δ^l is the vector with entries Δ_{jl}, and $\mathbf{u}^l$ is the unit vector with l-th entry 1, we can rewrite the above equation in matrix form:

$$\Delta^l = WD\Delta^l + \mathbf{u}^l.$$

Letting Δ be the matrix with entries Δ_{jl}, this gives us the equation:

$$\Delta = WD\Delta + I$$

$$\text{or} \quad (I - WD)\Delta = I.$$

Since $I - WD$ is a lower triangular matrix with unit diagonal it is non-singular and Δ is its inverse. Using this inverse for Δ, if $\mathbf{s}$ and $\mathbf{t}$ satisfy the equation $\mathbf{s}^T D\Delta = \mathbf{t}^T$, then

$$\mathbf{s}^T D = \mathbf{t}^T(I - WD).$$

As $(I - WD)$ is lower triangular, we can evaluate $\mathbf{t}$ from $\mathbf{s}$ by the equations

$$\mathbf{t}_i = f'(u_i)\left(\mathbf{s}_i + \sum_{(i,\ell)\in E} w_{\ell i}\mathbf{t}_\ell\right), \quad \text{for } i \in N, \tag{B}$$

in decreasing order of index, since $\mathbf{t}_\ell$ appears on the right hand side only if $\ell > i$.

This allows us to evaluate the partial derivatives of the outputs at individual nodes with respect to each weight in the network:

$$\begin{aligned}\frac{1}{o_k}\frac{\partial o_j}{\partial w_{lk}} &= f'(u_j)\frac{1}{o_k}\frac{\partial u_j}{\partial w_{lk}} \\ &= f'(u_j)\Delta_{jl} \quad \text{by (A)} \\ &= (\mathbf{u}^j)^T D\Delta\mathbf{u}^l.\end{aligned}$$

Using the above method of evaluation we can write

$$\frac{\partial o_j}{\partial w_{lk}} = o_k\mathbf{t}_l^j,$$

where by (B), $\mathbf{t}$ is given by the equations:

$$\mathbf{t}_i^j = f'(u_i)\left(\delta_{ij} + \sum_{\ell=i+1}^{n} w_{\ell i}\mathbf{t}_\ell^j\right) \quad \text{for } i \in N. \tag{Y}$$

With this general derivation we can derive the generalised delta rule of back-propagation as

$$\frac{\partial\|\mathbf{e}\|_2^2}{\partial w_{lk}} = 2o_k\mathbf{t}_l,$$

where $\mathbf{t}$ is given by:

$$\mathbf{t}_i = f'(u_i)\left(\mathbf{e}_i + \sum_{\ell=i+1}^{n} w_{\ell i}\mathbf{t}_\ell\right) \quad \text{for } i \in N.$$

and it is understood that the error vector $\mathbf{e}$ has been extended to have an entry for each node, the entries for non output nodes being 0.

3.2 Derivation of the Linear Programming Algorithm

As mentioned in the introduction 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 we suggest that individual errors should not be allowed to increase. If the weight changes are given by δw, we require that

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

We will take a local first order approximation to this inequality:

$$\sum_{(k,l)\in E} \frac{\partial (\mathbf{e}_j^p)^2}{\partial w_{lk}} \delta w_{lk} \leq 0$$

for all $j \in O$ and $p \in P$, where δw_{lk} is the small change made to the weight on the edge (k,l). The weight changes must be restricted in order to minimise second order effects:

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

where ϵ_{lk} are small non-negative quantities. Our learning requirement is to minimise some overal measure of error subject to the above constraints. Choosing the 2-norm and simplifying the expressions using the results of the previous section gives the following linear programme K:

$$\begin{aligned}
\text{Subject to:} \quad & \sum_{(k,l)\in E} -o_k^p \mathbf{t}_l^{pj} \delta w_{lk} \geq 0, \quad \forall p, j; \\
& \delta w_{lk} \geq -\epsilon_{lk}, \quad \forall (k,l) \in E; \\
& -\delta w_{lk} \geq -\epsilon_{lk}, \quad \forall (k,l) \in E; \\
\text{minimise} \quad & \sum_{(k,l)\in E} \left(\sum_{p\in P} o_k^p \mathbf{t}_l^p \right) \delta w_{lk}; \\
\text{where} \quad & \mathbf{t}_i^{pj} = f'(u_i^p) \left(\mathbf{e}_j^p \delta_{ij} + \sum_{(i,\ell)\in E} w_{\ell i} \mathbf{t}_\ell^{pj} \right); \quad \forall p, j, i; \\
\text{and} \quad & \mathbf{t}_i^p = \sum_{j\in O} \mathbf{t}_i^{pj} = f'(u_i^p) \left(\mathbf{e}_i^p + \sum_{(i,\ell)\in E} w_{\ell i} \mathbf{t}_\ell^p \right); \quad \forall p, i.
\end{aligned}$$

We should emphasise that the solution of the linear programme K 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. We should also discuss the difference between our use of a linear programme to solve local optimisation in a multi-layer network and the linear programme formulation of the single layer perceptron problem given in [Du-Ha73].

Duda and Hart use a linear programme to determine when a set of samples are not linearly separable, that is when the perceptron convergence algorithm will fail. They also give a linear programme formulation for choosing a good approximation when a separable (exact) solution does not exist. This suggests that the perceptron convergence theorem might be used to deliver an approximate solution to a general linear programme. In Section 5 we will propose this as a way of approximating our algorithm on large networks where it may be impractical to obtain an exact solution of the linear programme.

Setting $\delta w_{lk} = 0$ for all $(l,k) \in E$ is a (trivial) feasible solution of the LP and since the feasible region is clearly bounded there is always an optimal solution. This programme is more easily solved by transforming to the dual LP which we denote K^*:

$$\begin{aligned}
\pi_{pj} &\geq 0; \quad \forall p \in P, j \in O; \\
\pi^+_{lk} &\geq 0; \quad \forall (k,l) \in E; \\
\pi^-_{lk} &\geq 0; \quad \forall (k,l) \in E; \\
\sum_{p,j} \pi_{pj}(-o^p_k \mathbf{t}^{pj}_l) + \pi^+_{lk} - \pi^-_{lk} &= \sum_p o^p_k \mathbf{t}^p_l; \quad \forall (k,l) \in E; \\
\text{maximise} \quad \sum_{(k,l)\in E} &- \left(\pi^+_{lk} + \pi^-_{lk}\right) \epsilon_{lk}.
\end{aligned}$$

In order to investigate when the linear programme K has a non-trivial solution consider the $|P||O|$ vectors $\mathbf{x}^{pj}$ of dimension $|E|$ defined by

$$\mathbf{x}^{pj}_e = o^p_k \mathbf{t}^{pj}_\ell, \quad \text{where } e = (k,l) \in E.$$

We can characterise the behaviour of K in terms of these vectors as the following proposition demonstrates.

Proposition 3.2.1 : *The trivial solution to K is optimal if and only if there is a strictly positive linear combination of the vectors $\mathbf{x}_e$ equal to $\mathbf{0}$.*

Proof: If the trivial solution is optimal then the value of the optimal of the dual LP is also 0. Hence

$$\sum_{(k,l)\in E} -(\pi^+_{lk} + \pi^-_{lk})\epsilon_{lk} = 0,$$

and since $\epsilon_{lk} > 0$ and $\pi^+_{lk} + \pi^-_{lk} \geq 0$ for all $(k,l) \in E$, we have $\pi^+_{lk} = \pi^-_{lk} = 0$. In this case

$$\begin{aligned} \sum_{p,j} \pi_{pj}(-o^p_k \mathbf{t}^{pj}_l) &= \sum_p o^p_k \mathbf{t}^p_l \\ &= \sum_{p,j} o^p_k \mathbf{t}^{pj}_l, \\ \text{so} \quad \sum_{p,j} (\pi_{pj}+1) o^p_k \mathbf{t}^{pj}_l &= \sum_{p,j} (\pi_{pj}+1)\mathbf{x}^{pj}_e \\ &= 0, \quad \text{for all } e = (k,l) \in E. \end{aligned}$$

Since $\pi_{pj} + 1 > 0$ and we have a strictly positive linear combination of the vectors $\mathbf{x}^{pj}$ as required.

Now consider a strictly positive linear combination of the $\mathbf{x}^{pj}$ which sums to $\mathbf{0}$.

$$\sum_{pj} \eta_{pj} \mathbf{x}^{pj} = \mathbf{0}.$$

Choose $\lambda > 0$ such that $\lambda\eta_{pj} \geq 1$ for all p, j and set $\pi_{pj} = \lambda\eta_{pj} - 1 \geq 0$. Then since

$$\begin{aligned} \sum_{p,j} \pi_{pj}(-o^p_k \mathbf{t}^{pj}_l) &= \sum_{p,j} (\lambda\eta_{pj} - 1)(-o^p_k \mathbf{t}^{pj}_l) \\ &= -\lambda \sum_{p,j} \eta_{pj} \mathbf{x}^{pj}_e + \sum_{p,j} o^p_k \mathbf{t}^{pj}_l \\ &= 0 + \sum_p o^p_k \mathbf{t}^p_l, \end{aligned}$$

we can set $\pi^+_{lk} = \pi^-_{lk} = 0$ to obtain a feasible solution of K^* having a cost of 0. But the cost of K^* is

$$\sum_{(k,l)\in E} -(\pi^+_{lk} + \pi^-_{lk})\epsilon_{lk} \leq 0$$

and so the feasible solution we have found is also optimal, implying that the optimal value of K is also 0. Hence the trivial solution of the primary is an optimal solution. ∎

This proposition is very encouraging. Provided we have more edges than the product of the number of presentations and output nodes and that the presentations are not in

some sense degenerate, the vectors will not be linearly dependent. So in this case there will certainly not be a positive linear combination equal to zero, and so the linear programme will have a non-trivial solution.

It also gives an indication of how extra presentations, which are "similar" to existing ones, might be correctly processed if they were required to give the same output as the existing presentation. This is precisely generalisation. This will occur if the vector $\mathbf{x}^{pj}$ for the new presentation is in approximately the same "direction" as for the existing presentation (this will be our definition of "similar"). The new vector will be very unlikely to allow the creation of a positive linear combination equal to zero if one did not already exist. So addition of "generalised" presentations will not affect the existence of a non-trivial solution for the linear programme, at least in that locality of weight space. In this way a network with a particular weight assignment can be seen to confer on the space of possible presentations some similarity metric which may be used to characterise generalisation. This promises to be a very fruitful avenue for theoretical investigation of some of the more elusive properties of neural networks.

4 Experimental Findings

We have implemented a flexible simulation package which allows the user to interactively change between the standard back-propagation learning algorithm and the linear programming algorithm introduced above. Further it allows interactive manipulation of the network for experimentation. This section is intended to describe some of the experimental results obtained using this package.

Example 4.0.1 : *The neural network was required to recognise parallel line classes in the affine plane over the field of 2 elements.*

This problem was given to a network with 4 input nodes and 2 hidden nodes and one output node. If the 4 input nodes are arranged as a 2×2 grid then the first two presentations are the two horizontal lines and have output 0, the second two presentations are the two diagonal lines and have output 1 and the final two presentations are the vertical lines,

having again output 1. This problem was chosen as particularly difficult (although clearly very small) since it is a generalisation of the "exclusive or" problem.
Using the standard back-propagation algorithm (BP) we found that starting with random weights in approximately half of the learning trials the network converged to one which did not solve the problem. Using the linear programming algorithm (LP) all of the errors always tended to zero and the learning took far fewer iterations, typically 20 rather than 500 for BP.
In the case where BP fails to converge to a correct solution at least one error tends to one. In this case BP was allowed to run for 500 iterations until the large error was about 0.999992. At this point the LP algorithm was engaged for just 30 iterations. The largest error was reduced to about 0.998. The BP algorithm was now reengaged and all of the errors converged to zero.
Further experiments were performed by increasing the number of hidden nodes (with full connectivity) to 15. The LP algorithm solved the problem in even fewer iterations (though of course each iteration took several minutes of microvax time), while BP is still just as unreliable. ∎

Example 4.0.2 : *The neural network was required to distinguish between parallel line classes in the affine plane over the field of three elements.*

This problem was given to a network with 9 input nodes, 3 hidden nodes and one output node. There were in all 12 presentations corresponding to the 12 lines in the affine plane. These lines fall into four parallel classes each consisting of 3 lines. Two of the classes were required to have output 1 and two were required to have output 0. This problem is strictly more difficult than XOR because no pair of input nodes gives enough information to determine the output (order 3). Again the BP algorithm converged to incorrect solutions, but the LP algorithm always produced a correct solution. We found that, in this case also, LP can be used to "correct" the back-propagation process. ∎

Example 4.0.3 : *The Neural network was required to distinguish between digits on a 20 pixel retina.*

This problem was solved with two different network configurations. The first was a fully

connected network with 20 input nodes, 16 hidden units and 10 output nodes. All possible connections except those between input and output nodes were allowed, giving a total of 600 weights. The second network had two hidden layers containing 20 and 6 units respectively. Here the total number of weights was 580. There were 3 versions of each digit used in the experiment, giving 30 possible presentations, see Figure 1.

— Figure 1 about here —

Using back-propagation the problems of faulty convergence were experienced on some outputs even when only the first 10 of the 30 presentations were presented. With the Linear programming algorithm convergence with only 10 presentations was fast (about 15 iterations for the fully connected network and 38 for the layered network, with each iteration about 1.5 minutes of CPU on a VAXstation 3500 using the NAG library routine E04MBF to solve the linear programme).

When a further 10 presentations were added into the training set and training was continued, the layered network redressed the balance needing a further 25 iterations (iterations now took approximately 6 minutes of CPU) to converge while the fully connected network needed 42 iterations. The last 10 presentations were used to test generalisation with extremely noisy input. The fully connected network performed worse making a good guess for 5 of the 10 digits (1, 2, 3, 6, 9), while the layered network made a correct guess for 6 (3, 5, 6, 7, 8, 9). ∎

For a discussion of the significance of these results the reader is refered to the concluding section. The next section addresses the problem of implementing the linear programming algorithm efficiently.

5 Implementation Strategies for the Linear Programming Algorithm

The linear programming algorithm appears to exhibit impressive behaviour on small but difficult examples. Its efficiency on large examples is difficult to test because the size of the linear programme increases very rapidly with network size. For example, a fully connected network with 100 nodes (about 5000 edges) would require a matrix with over 50000000 entries to solve the linear programme. Apart from the space requirements it is also unclear how the time taken scales up with number of presentations and number of edges. The experimental experience was that the more critical dependency was on the number of constraints, that is the number of presentations times the number of outputs. Note that the linear programme should be solvable provided this number does not exceed the number of edges. When this number doubled from 100 to 200 in Example 4.0.3, the time taken for each iteration quadrupled.

One obvious economy to make is to combine the errors from the separate outputs of a presentation so that only one back-propagation is performed per presentation and only one constraint is added. Though this will allow processing of a larger number of presentations, experience with Example 4.0.3 showed that the same problem occurred as with standard back-propagation, namely one or more individual output errors increased, though the overall error for the presentation decreased.

Thus it would seem that for larger networks both the space and time constraints make the linear programming algorithm intractable. In this section we discuss ways in which the algorithm or approximations to it might be used on larger problems and networks. We will mention three techniques, presenting them in increasing order of adherence to the original Linear Programming algorithm.

Approach 5.0.1 : *Monitored Back-propagation*

A first approach follows only loosely the spirit fo the original requirement put on the linear programming algorithm, namely that at each iteration all of the errors are reduced. Standard back-propagation is applied to the network and the individual errors are monitored. All individual errors which increased during an iteration are recorded. The presentations with these increased errors would be back-propagated at the next several iterations until

the errors were reduced to their original values. Once this had been achieved (this should be possible provided the original step size was sufficiently small), global iterations could be continued. We call this method monitored BP. ∎

Approach 5.0.2 : *Perceptron Linear Programming*

The second approach is more faithful to the algorithm and involves replacing the exact solution to the linear programme with an approximate one. It involves an application of the perceptron convergence theorem of Rosenblatt [Ro59]. For this reason it will be called Perceptron LP. The constraints of the linear programme define hyperplanes:

$$\sum_{(k,l)\in E} -o_k^p \mathbf{t}_l^{pj} \delta w_{lk} \geq 0, \quad \forall p, j,$$

and solving the linear programme involves finding a weight vector which satisfies all of these inequalities. But this is precisely what the perceptron convergence theorem does, provided there exists some $\eta > 0$ and a unit weight vector satisfying

$$\sum_{(k,l)\in E} -o_k^p \mathbf{t}_l^{pj} \delta w_{lk} \geq \eta, \quad \forall p, j.$$

This requirement is stronger than that required for solution of the linear programme, since there may exist non-trivial solutions which leave some errors unchanged. For practical purposes, however, it will probably be sufficient to require that errors do not increase by more than some $\eta > 0$ and convergence can be expected even if the second inequalities above cannot be satisfied. Once a solution of the inequalities has been found using the perceptron convergence algorithm, we can multiply it by a scalar factor so that for each component δw_{lk} we have $-\epsilon_{lk} \leq \delta w_{lk} \leq \epsilon_{lk}$. The resulting weight change vector will be a feasible solution of the linear programme and so will reduce each of the individual errors, thus resulting in an overall reduction in error. Naturally, this solution will not generally be the optimal solution of the Linear Programme.

In this application the perceptron convergence algorithm initialises the weight change to some value. The obvious choice in this case is that given by the standard back-propagation algorithm. We then iterate through the presentations (leaving the network weights unchanged but updating the proposed changes) and test the predicted change in the errors from the proposed changes. If for any presentation the predicted change is that

the error increases the derivatives corresponding to that presentation are added into the proposed change. The iteration terminates when the predicted change for all the presentations is a reduction in the error. At this point the normalisation is performed and the weight change implemented.

In order to save memory in a large simulation we do not expect to have all the derivatives available in memory. By retaining the values $\mathbf{t}_l^{pj}$ at each node we could significantly reduce the storage required. If this is still too large a requirement, we could process a subset of presentations. As a presentation came to satisfy the error reduction requirement it would be removed from the set and an as yet unprocessed presentation could replace it chosen using some fairness criterion. When all the presentations have been processed we cycle through them once more to verify that subsequent changes have not affected the earlier presentations. In this way we could effect a significant reduction in storage without recomputing the derivatives each time a presentation is considered.

By accepting an approximate solution to the linear programme we have solved the problem of the very large storage requirements of the standard linear programme solvers. There is very little known about the time complexity of the perceptron convergence algorithm, so that it is not clear if this technique is practical for large networks in terms of time complexity. ∎

Approach 5.0.3 : *Structured Learning*

The third approach to overcoming the time and space constraint of the large linear programme is to present only a subset of all the presentations at each iteration. This set might be selected by some random distribution or some grouping criterion designed to place potentially conflicting inputs together. The linear programming algorithm would then be applied to this subset using only a subset of the weights as variables. The subset of weights that is used could again either be chosen at random or some account could be taken of the relative importance of weights to the chosen presentations, probably measured by the relative sizes of their derivatives.

The third technique can be given psychological justification as learning in context. When biological brains learn new concepts they are usually learnt by comparing and distinguishing with known objects. The structured learning suggested here applies that same principle to Neural Network learning.

Clearly the problem of size and time complexity is overcome, but at the expense of introducing a great many indeterminates as to how presentations and edges are selected. The overall convergence is also further obscured. It seems, however, to be a fruitful avenue of investigation, particularly in relation to introducing structure into the presentation inputs. ∎

6 Conclusions

Back-propagation is a very effective and useful learning scheme for feedforward networks. It does however suffer from being unpredictable. The main aim of this paper has been to develop a similar (in the sense of using gradient descent) learning algorithm which is more robust.

The linear programming algorithm appears to provide rapid convergence (in terms of the number of iterations) and to converge reliably to a network that has learnt the presented stimulus response pairs. The learning also appears to be better distributed over the network resulting in the network having better degradation properties.

Using the LP algorithm it may be possible to learn more complicated problems with a given network than is possible using back-propagation. This is because one solution to the convergence failure of back-propagation is to increase the size of the network. In general it is desirable to solve a problem with a network which is close to the minimum size, since this will increase the chances of good generalisation.

The major criticism of the proposed linear programme algorithm is the time taken to execute each iteration and the practicalities of execution for very large networks. This criticism is, however, put in perspective when for example a three layer network set to solve the XOR problem took 29 LP iterations as opposed to 12 million BP iterations. We have presented three methods, the Monitored BP, the Perceptron LP and the Structured Learning methods, all of which approximate the linear programming algorithm and which would be practical for larger networks.

A comparison of these methods should be of interest as the first method emphasises

the non-increasing of errors aspect of the linear programming algorithm without doing work to disambiguate the presentations, while the second method disambiguates the errors but probably does not retain the large weight changes in all parts of the network typical of standard LP. The last approach manages all aspects of LP but creates the problems of how to group presentations and weights. This is in contrast to standard back-propagation which typically moves weights in later layers of the network by greater amounts than those in earlier layers.

References

[Du-Ha73] : Duda, Richard O. and Hart, Peter E. (1973). *Pattern Classification and Scene Analysis.* John Wiley and Sons.

[He88] : Hecht-Nielsen, R. (1988). Theory of the Backpropagation Neural Network. *Abstracts of 1st Meeting of INNS.*

[Kr-Me87] : Krauth, W. and Mezard, M. (1987). Learning algorithms with optimal stability in neural networks, *Journal of Physics A: Mathematical and General*, **20**, L745–751.

[Mc-Ru86] : McClelland, J. L. and Rumelhart, D. (1986). *Parallel Distributed Processing, Vols 1 and 2.* MIT Press.

[Mi-Pa88] : Minsky, M. and Papert, S. (1988). *Perceptrons, expanded edition.* MIT Press.

[Ro59] : Rosenblatt, Frank. (1959). Two Theorems of Statistical Separability in the Perceptron. In *Proceedings of a Symposium on the Mechanisation of Thought Processes* (pp. 421–456). Her Majesty's Stationary Office: London.

[Ru-Hi-W86] : Rumelhart David E., Hinton, Geoffrey E. and Williams, Ronald J. (1986). Learning representations by back-propagating errors. *Nature,* 323.