AN ACCELERATED ERROR BACK-PROPAGATION LEARNING ALGORITHM

by S. MAKRAM-EBEID, J.-A. SIRAT and J.-R. VIALA

Laboratoires d'Electronique Philips, 3 avenue Descartes, B.P. 15, 94451 Limeil-Brévannes Cédex, France

Abstract

We propose a method for learning in multilayer perceptrons (MLPs). It includes new self-adapting features that make it suitable for dealing with a variety of problems without the need for parameter re-adjustments. The validity of our approach is benchmarked for two types of problems. The first benchmark is performed for the topologically complex parity problem with a number of binary inputs ranging from 2 to 7. We reduce the learning times by two to three orders of magnitude compared with conventional error back-propagation (EBP). The second problem type occurs when a high accuracy in separating example classes is needed. With classical EBP techniques and even for a one-dimensional input the learning time sharply increases with decreasing interclass Euclidean distances $\varepsilon$ according to a law of the form $1/\varepsilon^2$. Our algorithm yields substantially shorter learning times that behave as $\log(1/\varepsilon)$. We demonstrate satisfactory learning for problems combining topological and accuracy difficulties and for which conventional EBP is practically useless.

Keywords: adaptive algorithms, multilayer perceptrons, neural networks, performance optimization, supervised learning

1. Introduction

Multilayer perceptrons (MLPs) are among the most promising types of neural nets and have given rise to a large number of applications as witnessed by recent conferences on the subject (see for example ref. 2). The discovery and popularization of the error back-propagation (EBP) algorithm has strongly stimulated research on MLPs and on their applications. However, the original form of the EBP often leads to very long learning times.

Following the analysis of Lippmann, each neuron in the input layer of an MLP divides the input vector space into simple half-space regions. The other neuron layers perform logical combinations of these half-spaces to yield...
S. Makram-Ebeid, J.-A. Sirat and J.-R. Viala

topologically more complex decision regions. The learning difficulties that can be encountered consist either in providing complex logical function mappings or in separating decision regions which are 'close' in the sense of Euclidean distances. The second type of learning difficulty is particularly important when one deals with continuously coded input variables.

Parity is a logical mapping problem which is reputed to be difficult to implement on an MLP. It consists in presenting a two-layer perceptron with a number $N$ of binary inputs and requiring an ON output when the number of ON inputs is odd and an OFF output otherwise. New adaptive EBP techniques have been proposed to solve this type of 'logical mapping' problem. A review of these techniques and of their relative merits has been presented recently by Jacobs.

For the practical applications (pattern recognition) related to those which we study in our laboratory, the second type of learning difficulty mentioned above was found to be important when continuously coded inputs were used. This becomes particularly critical if one has to separate example classes for which the inputs differ very slightly. The use of an adaptive form of EBP reduces learning times, although not by orders of magnitude.

In this paper, we develop a new algorithm that brings substantial improvements in learning performance. Previous publications have concentrated on the exclusive OR problem to benchmark their learning approaches. In contrast, we tested our algorithm on the two problem categories we identified above. The first category is represented here by the parity problems. We study the effect of increasing complexity when the number of binary inputs grows from 2 (exclusive OR) to 7. We describe such problems as topologically complex. We require from our algorithm that it automatically adapts to these topologically complex problems as well as to the second problem category where accuracy difficulties are encountered. We attempt to achieve these goals in such a simple way that the algorithm can be implemented on parallel neurocomputing hardware.

2. Description of the new algorithm

Let us first define our nomenclature for the parameters in an MLP. The number of neuron layers is represented by $L$ and we assume there are $I(l)$ neurons in the $l$th layer, each of these being referenced by the double label $(i, l)$ with $i$ ranging from 1 to $I(l)$. The net input to any such neuron is thus represented by $x_{i,l}$ and the corresponding output potential by $y_{i,l}$. In particular, the input variables to the MLP are denoted by $y_{l,0}$ and correspond to $l = 0$. 
The different neuron potentials in the MLP are related by

\[ x_{j,l} = \left( \sum_{i=1}^{L} w_{ij,i-1} y_{i,i-1} \right) - s_{j,l} \quad \text{for } 1 \leq l \leq L \quad (1) \]

where \( s_{j,l} \) denotes a threshold and \( w_{ij,i-1} \) is a synaptic coefficient. The input and output potentials of the neurons are related by the sigmoid response of the neurons in the form

\[ y_{j,l} = \tanh(x_{j,l}/T_l) \quad (2) \]

where we give all neurons in the layer \( l \) the same temperature \( T_l \).

In the following we describe the various steps involved in our algorithm.

2.1. Gradient back-propagation

On presentation of any example labelled \( p \) from the example base to the MLP, we let the net evaluate the corresponding output potentials \( y_{i,L} \). We introduce a new definition for the partial error (or partial energy) \( E_p \) for a single example as

\[ E_p = \frac{1}{2} \sum_{i=1}^{L} \theta_{i,L}(y_{i,p}^{(p)} - y_{i,L})^2 \quad (3a) \]

where \( y_{i,p}^{(p)} \) is the required response for example \( p \) and output neuron \( i \). This is the usual quadratic error expression with extra factors \( \theta_{i,L} \). These are meant to have an accelerating influence by favouring the learning of correct sign patterns before accurate output responses are required. To do so, we take

\[ \theta_{i,L} = 1 \text{ if } y_{i,p}^{(p)} y_{i,L} \leq 0 \quad \text{and} \quad \theta_{i,L} = \theta^+ \text{ if } y_{i,p}^{(p)} y_{i,L} > 0 \quad (3b) \]

where \( \theta^+ \) is a quantity between 0 and 1 that we gradually bring nearer to unity for successive learning steps. As will be seen later, a similar sign-dependent strategy will be generalized to all neuron layers. With these definitions in mind, we can now compute the different components \( g_{i,l} \) of the gradient of the partial error \( E_p \) with respect to the different neuron input potentials \( x_{i,l} \) as defined by

\[ g_{i,l} = \frac{\partial E_p}{\partial x_{i,l}}. \quad (4) \]
An efficient method for the calculation of all these components is the gradient back-propagation algorithm \(^3\) namely

\[ g_{i,L} = \theta_{i,L}(y_{i,L} - y_p)\dot{f}_{i,L} \quad \text{for } i = 1, 2, \ldots, I(L) \]  

and

\[ g_{i,l} = \left( \sum_{j=1}^{\ell+1} w_{i,l} g_{j,l+1} \right) \dot{f}_{i,l} \quad \text{for } i = 1, 2, \ldots, I(l) \text{ and } l = L - 1 \text{ to } 1 \]  

where \( \dot{f}_{i,l} = \frac{\partial \tanh(x_{i,l}/T_l)}{\partial x_{i,l}} \).

### 2.2. A conjugate gradient with minimal disturbance

Classical EBP learning amounts to decreasing each neuron input potential by an amount \(-\Delta x_{i,l}\) proportional to the corresponding gradient component \(g_{i,l}\). The proportionality constant or gain factor must, in this case, be adjusted experimentally. A more quantitative approach is to specify a required decrease \(\eta E_p\) in the partial error \(E_p\) \(^4,7\) with the numerical factor \(\eta\) set to a value between 0 and 2. As for the Adaline learning rule \(^8\), this correction should be introduced with minimal disturbance of the neural net. In this way, partial learning for a given example \(p\) should minimally disturb the learning already achieved for other examples. We propose to measure quantitatively the disturbance introduced by such a partial learning by a Euclidean norm in the form

\[ D_1 = \sum_{i,l} \frac{|\Delta x_{i,l}|^2}{\beta_{i,l}} \]  

The different positive weighting factors \(1/\beta_{i,l}\) in the above expression are a measure of the relative importances of the different neurons in perturbing the MLP. We wish to reduce the partial error by an amount proportional to its current value \(E_p\) while keeping the disturbance measure \(D_1\) minimal. The correction introduced can be expressed linearly in terms of the changes \(\Delta x_{i,l}\) in the different net inputs to the neurons. There are therefore infinitely many solutions that will lead to a predetermined change in the error \(E_p\). Among these, we take the particular solution that minimizes the quadratic function \(D_1\). This is a simple problem that amounts to taking the pseudo-inverse of a vector \(^9\). The corresponding changes required in neuron net input potentials can be
expressed as

\[ \Delta x_{i,t} = -2\eta E_p \frac{\beta_{i,t} g_{i,t}}{G^2} \]

where \( G^2 = \sum_{i,t} \beta_{i,t} g_{i,t}^2 \). (7)

In the above expression, we wrote the adjustable gain factor in the form \( 2\eta \) to take into account the quadratic nature of \( E_p \) near its minimum when expressed in terms of the increments \( \Delta x_{i,t} \). The weighting factors we have introduced are defined by

\[ \beta_{i,t} = \beta (T_l \prod_{m=1}^{L} T_m) \theta_{i,t} \] (8)

where \( \theta_{i,t} = 1 \) if \(-g_{i,t} x_{i,t} \leq 0\) and \( \theta_{i,t} = \theta^+ \) if \(-g_{i,t} x_{i,t} > 0\).

The above definition of \( \theta_{i,t} \) can easily be seen to reduce identically to that of \( \theta_{i,L} \) in eq. (3b) for \( l = L \). The first factor \( \beta_l \) appearing in eq. (8) is positive and depends only on the layer number. It is given decreasing values when \( l \) increases from \( l = 1 \), for the first hidden layer, to \( l = L \) for the output layer (i.e. we take \( \beta_k > \beta_l \) whenever \( k < l \)). This is desirable because a disturbance in a layer near the output will modify the response of the MLP to all the examples and will, therefore, have larger consequences than a disturbance in a layer nearer to the input. Hence, slower learning for the near-output layers is needed. We always normalize to unity the sum of the factors \( \beta_l \) for all the layers.

The temperature-related term in eq. (8) ensures that, when the neuron sigmoids operate in the linear regions \((x_{i,t} < T_l)\), the corrections \( \Delta x_{i,t} \) introduced are in proportion to the corresponding temperature.

The factor \( \theta_{i,t} \) introduced for all layers in eq. (8) is a generalization of the sign-related term appearing in eqs (3b) and (5a) for the output layer. We take \( \theta_{i,t} = 1 \) if \( x_{i,t} \) and \( \Delta x_{i,t} \) are of different signs (recalling that \( \text{sign}(\Delta x_{i,t}) = \text{sign}(-g_{i,t}) \)). Otherwise, if \( \text{sign}(\Delta x_{i,t}) = \text{sign}(x_{i,t}) \), we take \( \theta_{i,t} = \theta^+ \). This will privilege learning through changing the net input signs and/or through reducing their magnitudes \( |x_{i,t}| \). In early learning stages, \( \theta^+ \) starts at a small positive value which gradually increases in later stages. Hence progressively more accuracy in the neuron responses is required.

2.3. Introducing a bounded conjugate gradient

The conjugate gradient formula of eq. (7) has an obvious snag since the amplitudes of the corrections can become very large if the modulus \( G \) of the
S. Makram-Ebeid, J.-A. Sirat and J.-R. Viala

gradient happens to be small. To avoid the resulting numerical instabilities, it would be natural to add a constant positive term to $G^2$ in the denominator of the right-hand side of eq. (7). We have tried this solution but found that often the only significant components $g_{i,l}$ correspond to small values of $\beta_{i,l}$ so that very slow learning results. We avoid these difficulties by adopting a modified conjugate gradient formula in the form

$$\Delta x_{i,l} = -2\eta E_p \frac{\beta_{i,l} g_{i,l}}{G^2 + \gamma(\bar{\beta} G^2)^{\nu/2}}$$

(9)

where $\gamma$ is a constant, $\nu$ is an exponent close to unity ($0 \leq \nu \leq 1$) and $\bar{\beta}$ is a weighted average value for the different coefficients $\beta_{i,l}$ defined by

$$\bar{\beta} = \frac{\left( \sum_{i,l} \beta_{i,l}^2 g_{i,l}^2 \right)}{\left( \sum_{i,l} \beta_{i,l} g_{i,l}^2 \right)}.$$

It can be seen that the extra term $\gamma(\bar{\beta} G^2)^{\nu/2}$ which we have added to $G^2$ in the denominator of eq. (9) automatically imposes an upper bound to the correction amplitudes. Simple calculus and algebra show that, if we take $\gamma$ in the form

$$\gamma = \left[ \frac{(1 - \nu)^{(1 - \nu)/(2 - \nu)}}{(2 - \nu)\xi T_1} \right]^{2 - \nu},$$

(10)

the upper bound to the corrections' moduli will be proportional to the temperature $T_1$ of the first layer:

$$\sqrt{\sum_{i,l} (\Delta x_{i,l})^2} \leq 2\eta E_p \bar{\beta}^{(1 - \nu)/(2 - \nu)} \xi T_1,$$

$\zeta$ being a 'correction modulus parameter' kept constant during learning. With these formulae and with $\nu$ close to unity, learning can be effective even for small values of $\bar{\beta}$ or $G^2$.

2.4. Modifying thresholds and synaptic coefficients with normalization constraint

Up to this point, we have defined the required changes in the net inputs of the MLP neurons. We now have to realize these changes as modifications in the synaptic coefficients and thresholds. The net input to an arbitrary neuron

526
Philips Journal of Research Vol 44 No. 6 1990
as defined in eq. (1) can be written in terms of a scalar product of two vectors as

\[ x_{j,l} = W_{j,l-1} \cdot Y_{l-1} - s_{j,l} \]  

(11)

where the components of the vectors \( W_{j,l-1} \) and \( Y_{l-1} \) are defined as \( w_{ij,l-1} \) and \( y_{i,l-1} \) respectively with \( i \) ranging from 1 to \( I(l-1) \). The region of the vectorial space to which \( Y_{l-1} \) belongs and for which the net input to neuron \((j, l)\) vanishes is a hyperplane and the region for which the corresponding sigmoid response is linear is defined by \(|x_{j,l}| < T_l\). This region can be geometrically described as a ‘slab’ of width \( 2T_l/\|W_{j,l-1}\| \). Learning mainly occurs when the sigmoids operate near their linear region. To keep a good control of this linear operating region we are therefore led to impose a unit modulus to the vector \( W_{j,l-1} \) during learning and to monitor the temperature \( T_l \) during the iterations. When this normalization is carried out, eq. (11) can be rewritten in the form

\[ x_{j,l} = \|Y_{l-1}\| \cos \phi - s_{j,l}, \]  

(12)

\( \phi \) being the angle between the two vectors \( W_{j,l-1} \) and \( Y_{l-1} \). Any change \( \Delta x_{j,l} \) in the net input \( x_{j,l} \) can therefore be decomposed into a partial change \( \Delta(\phi)x_{j,l} \) due to a change in the angle \( \phi \) and a change \( \Delta(s)x_{j,l} \) due to a change in the threshold \( s_{j,l} \). We obtain the most suitable decomposition of \( \Delta x_{j,l} \) by applying once again the principle of minimal perturbation. We take here, as a measure of the perturbation which should be minimized, the quantity

\[ D_2 = \sigma_{l-1}^2(\Delta \phi)^2 + (\Delta s_{j,l})^2. \]  

(13)

For the first hidden layer \((l = 1)\), we define the quantity \( \sigma_{l-1}^2 = \sigma_2^2 \) as the mean-square of the components of the input variables for the example base. For the upper layers, we set \( \sigma_{l-1}^2 \) at a fixed value of 1. The choice of a polar perturbation measure (eq. (13)) is logical since \( \Delta \phi \) represents an angle of rotation of the hyperplane defined by \( \{x_{j,l} = 0\} \) in the vectorial space to which vector \( Y_{l-1} \) belongs, while \( \Delta s_{j,l} \) represents a translation of this hyperplane. By linearizing the expressions of \( \Delta x_{j,l} \) in \( \Delta \phi \) and \( \Delta s_{j,l} \), we obtain the decomposition \( \Delta x_{j,l} = \Delta(\phi)x_{j,l} + \Delta(s)x_{j,l} \) that minimizes the perturbation measure \( D_2 \) (eq. (13)) in the form

\[ \Delta(\phi)x_{j,l} = \Delta x_{j,l} \frac{\|Y_{l-1}\|^2 - (x_{j,l} + s_{j,l})^2}{\|Y_{l-1}\|^2 - (x_{j,l} + s_{j,l})^2 + \sigma_{l-1}^2} \]
and

\[ \Delta(t)_{j,l} = \Delta x_{j,l} - \Delta(\phi)_{x_{j,l}}. \]  

(14)

The normalization condition \( \| W_{j,l-1} \| = 1 \) causes the value of \( \Delta(\phi)_{x_{j,l}} \) to have an upper bound of \( \| Y_{l-1} \| - (x_{j,l} + S_{j,l}) \) and a lower bound of \( -\| Y_{l-1} \| - (x_{j,l} + S_{j,l}) \). Because eq. (14) has been obtained using linear approximations, it may lead to corrections violating these bounds. Therefore, if eq. (14) yields a value for \( \Delta(\phi)_{x_{j,l}} \) outside these bounds, we simply set it equal to the nearest bound and express \( \Delta(t)_{x_{j,l}} \) as the difference \( \Delta x_{j,l} - \Delta(\phi)_{x_{j,l}} \).

We can thus evaluate the required modification in threshold potential \( \Delta s_{j,l} = -\Delta(t)_{x_{j,l}} \).

From the above discussion, the only change in the weights that we have to consider can be geometrically treated as a rotation of the weight vector in the plane defined by the two \( I(l-1) \)-dimensional vectors \( Y_{l-1} \) and \( W_{j,l-1} \) (weight vector prior to change). The change in the synaptic vector \( \Delta W_{j,l-1} \) can thus be written as a linear combination of \( W_{j,l-1} \) and \( Y_{l-1} \) in the form

\[ \Delta W_{j,l-1} = A_{j,l} W_{j,l-1} + B_{j,l} Y_{l-1}. \]

(15)

We impose now the two conditions

\[ \Delta W_{j,l-1} \cdot Y_{l-1} = \Delta(\phi)_{x_{j,l}} \]

and

\[ \frac{1}{2} \Delta \| W_{j,l-1} \|^2 \simeq \Delta W_{j,l-1} \cdot W_{j,l-1} = 0 \]

which allows us to determine the two unknowns \( A_{j,l} \) and \( B_{j,l} \) in the form

\[ A_{j,l} = -\frac{\Delta(\phi)_{x_{j,l}} (x_{j,l} + S_{j,l})}{\| Y_{l-1} \|^2 - (x_{j,l} + S_{j,l})^2} \]

and

\[ B_{j,l} = \frac{\Delta(\phi)_{x_{j,l}}}{\| Y_{l-1} \|^2 - (x_{j,l} + S_{j,l})^2}. \]

(16)

2.5. Description of the learning iterations

We start the learning procedure by randomly initializing the weights and normalizing the modulus of each of the corresponding vectors \( W_{j,l-1} \) to unity. In accordance with the terminology used in the literature \(^3\), we define a learning
Accelerated EBP learning algorithm

epoch as a sequence where each example of the base is presented once and only once to the MLP. Each of these example presentations is referred to as a learning iteration. Within an iteration, if the partial error $E_p$ for an example $p$ is found smaller than $10^{-4}$, we do not introduce any correction. Otherwise, the error correction procedure described in the previous sections is applied. This procedure is summarized as follows:

1. we determine the partial error (eq. (3)) and the corresponding gradient components with respect to the different neuron net input potentials (eq. (5));
2. we compute the corresponding weighting coefficients $\beta_{i,t}$ (eq. (8)) and the required change in net inputs (eqs (9) and (10));
3. these changes in net inputs are then translated into changes in threshold potentials and synaptic coefficients (eqs (14)–(16)).

In view of the linearizing approximations used and of the computational rounding errors, the moduli of the synaptic vectors $W_{j,l-1}$ do not remain strictly equal to unity. In practice, we find it sufficient to renormalize these vectors once for each complete learning epoch. The order of presentation of all the examples of the base within each epoch is chosen randomly. The temperatures of the sigmoids in the different layers are decreased geometrically after each completed learning epoch, namely, we set

$$T_l = T_l^{(0)} \rho_l^{-1}$$

where $T_l^{(0)}$ is the initial temperature in layer $l$, $\rho_l$ is a geometrical reduction factor (smaller than unity) and $n$ is the epoch order number. The sign-sensitive factor $\theta^+$ appearing in the definition of the quantities $\theta_{i,l}$ in eqs (3) and (8) is determined from the formula

$$\theta^+ = \frac{T_c}{(T_c + T_1)}$$  \hspace{1cm} (17)

where $T_c$ is a constant 'critical' temperature. At the start of iterations, the input layer temperature $T_1$ is large and the corresponding value of $\theta^+$ is small. For subsequent iterations, as $T_1$ decreases exponentially, $\theta^+$ grows nearer to unity.

The iterations are stopped either successfully, when all the examples are learned, or unsuccessfully when the input temperature has reached a preset value $T_{\text{min}}$ below which one considers further learning to be practically impossible.
3. Evaluation of algorithmic performance

We have carried out sequential-computer simulations to study the validity of the concepts proposed in this paper. Each one of the simulation results presented was obtained by averaging over 100 different randomly chosen initial settings of the MLP's synaptic weights and thresholds. For every initial setting, a learning 'trial' was performed. The corresponding percentage \( P(\text{success}) \) of those trials leading to complete learning of all the examples was computed in each case. We define the effective number of learning epochs \( K_{\text{eff}} \) as the average total number of epochs for successful and unsuccessful trials needed to obtain one satisfactory learning. The value of \( K_{\text{eff}} \) was calculated from the total number of learning epochs divided by the fraction of successful trials \( (P(\text{success})/100) \). Our measure of learning time is the effective number of iterations calculated from the product of \( K_{\text{eff}} \) with the number of examples in the base. Of course, some uncertainty is incurred in these estimates whenever \( P(\text{success}) \) is less than 100%. This is because the total number of epochs depends on the minimum temperature \( T_{\text{min}} \) used as a condition for stopping unsuccessful learning sequences. In this work, we set \( T_{\text{min}} \) to one-quarter of the minimal Euclidean distance between the example classes. In general, one does not have an accurate knowledge of this Euclidean distance but an order-of-magnitude estimate will be enough to determine a sufficiently small value of \( T_{\text{min}} \) that avoids a premature interruption of the iterations. In practice, an underestimation of \( T_{\text{min}} \) by an order of magnitude will not cause an appreciable increase in learning times.

As explained in Sec. 1, we have benchmarked our algorithm for two types of problems. A two-layer perceptron with a single output was required. For each example base, we carried out systematic learning statistics for different combinations of the learning parameters of our algorithm. The best combination of learning parameters was found to be nearly the same for the different problem types, namely

\[ \eta = 1.2, \quad \zeta = 0.65/T_1^{(0)} \quad \text{and} \quad v = 0.9. \]

Near-optimum learning times were obtained with a ratio of gains of the two layers \( \beta_1/\beta_2 \) set to any value between 3 and 10 times the number of examples in the base. The parameter \( T_c \) must be set to a value between one and two orders of magnitude smaller than the minimum Euclidean distance between the example classes. The algorithm proved to be tolerant to rather large variations in the parameter values as illustrated in the next section.
3.1. Results for the parity problem

In the parity problem, we took each of the \( N \) inputs to be either \(+1\) or \(-1\). Unless otherwise stated, the corresponding MLP had \( N \) inputs, \( N \) neurons in the hidden layer and a single output neuron (\( N/N/1 \) MLP architecture). The output was required to respond with a positive sign for an odd number of \(+1\)s inputs and with a negative sign otherwise. All the \( 2^N \) possible binary combinations were taken to form the example base. These examples can be grouped into subclasses, each corresponding to a fixed number of \(+1\)s in the input. Geometrically, these subclasses are located in hyperplanes. The minimum Euclidean distance between these hyperplanes can be calculated as \( 2/j_0 \). In accordance with the previously defined rules, we took \( T_{\text{min}} = 0.5/\sqrt{N} \) and \( T_c = 0.1/\sqrt{N} \). Optimal learning times corresponded to the general parameter setting defined above together with \( T_1^{(0)} = T_2^{(0)} = 0.55 \) and \( \rho_1 = \rho_2 = 0.999 \) (see Sec. 2.4 for the definition of \( \rho_1 \)).

3.1.1. Illustrating the role of the learning parameters for five-input parity

As mentioned above, we have carried out a systematic simulation study of the effect of the different parameters of our algorithm for parity- and accuracy-related problems. We compared the sensitivities of our algorithm to parameter variations for these two problem categories. A higher sensitivity was observed for the parity problems but even then, as illustrated in fig. 1, our algorithm showed remarkable tolerance to wide parameter changes. In spite of this tolerance, all the distinctive features of the algorithm are seen to play a significant role in improving the learning performance. In each part of fig. 1 we study the effect of varying a single parameter of the algorithm while keeping the others at their optimal values. The case presented corresponds to learning for the five-input parity problem (\( N = 5 \)). This is considerably more complex than the exclusive OR (two inputs parity) problem. The corresponding number of examples was 32. Very similar results were obtained for the other example bases studied. The square symbols correspond to the percentage \( P(\text{success}) \) of successful trials. The circles correspond to the number \( K_{\text{conv}} \) of learning epochs to convergence averaged over all successful trials. The triangles in fig. 1a) represent the effective number \( K_{\text{eff}} \) of learning epochs which, for a fixed number of examples serves as a measure of learning time. In fact, the minimum value of \( K_{\text{eff}} \) was always found to correspond nearly to the minimum of the ratio \( K_{\text{conv}}/P(\text{success}) \).

In Fig. 1a) we study the effect of varying the parameter \( \rho_1 \) (geometric reduction ratio for the temperature of the first layer). Practically 100% successful trials are obtained for \( 1 - \rho_1 \leq 0.01 \) but the shortest learning time
Fig. 1. Parametric study of our algorithm for five-input parity. Every result displayed is an average over 100 randomly chosen initial settings of the synaptic coefficients and thresholds: □, success percentage \( P(\text{success}) \); ○, average number of learning epochs to convergence, \( K_{\text{conv}} \); △, effective number of epochs, \( K_{\text{eff}} \). a) Influence of the parameter \( \rho_1 \); b) influence of the ratio \( \beta_1/\beta_2 \); c) the effect of the exponent \( v \); d) the effect of \( T_c \).

does not correspond to \( P(\text{success}) = 100\% \). It is noteworthy that, as \( 1 - \rho_1 \) increases between 0.01 and 0.3, both the success percentage \( P(\text{success}) \) and the average number \( K_{\text{conv}} \) of learning epochs decrease but the corresponding effective number of learning epochs remains roughly constant. The shortest learning time is obtained with \( 1 - \rho_1 \approx 0.1 \) with a corresponding success percentage below 50%. Qualitatively similar observations have been made for all the cases we have studied.

Figure 1b) shows the influence of the ratio \( \beta_1/\beta_2 \) between the gains of the first and second layers of the MLP and demonstrates the advantage of working with a large value of this ratio.

Figure 1c) demonstrates the advantage of working with a near unity exponent \( v \) in the conjugate gradient expression (eq. (9)) for the corrections in net inputs.
The importance of the parameter $T_c$ is illustrated in fig. 1d. It is recalled that $T_c$ enters the definition (eq. (17)) of $\theta^+$ which is, in turn, used in defining the sign-dependent gain factors $\theta_{i,i}$ in eqs (3) and (8). For the parity problems, the optimum value of this parameter proves to be about one-tenth of the minimum Euclidean distance between the example classes. When $T_c$ decreases below this optimum value, the success percentage $P(\text{success})$ remains equal to 100% but the average number of epochs increases. In comparison, for the accuracy-related problems, both $P(\text{success})$ and the number of epochs needed remain practically constant when $T_c$ decreases below this optimum value. Working with values of $T_c$ larger than the optimum value is never advisable because the success percentage $P(\text{success})$ decreases and the average number of epochs increases.

3.1.2. Benchmarking our algorithm for parity problems

Figure 2 shows the result of a comparison of our algorithm with those of classical EBP for the parity problems. The results for the classical EBP were optimized by determining for each parity problem ($N = 2$ to 5) the values of the gain and momentum coefficient $\alpha^3$ which minimize the learning time. On the ordinate, the learning time is defined as the average total number of example presentations (iterations) needed to achieve one complete learning of the base. We gain about three orders of magnitude in learning time for an odd number of binary inputs ($N = 3$ and 5) and about two orders of magnitude for an even number of inputs ($N = 2$ and 4). In contrast to conventional EBP,
the parity problems with 6 and 7 inputs prove quite tractable with our method. Learning with our algorithm was achieved with success percentages of $P(\text{success}) = 100\%$ for $N \leq 5$, $P(\text{success}) = 40\%$ for $N = 6$ and $P(\text{success}) = 93\%$ for $N = 7$ (for 100 trials in each case). The fluctuations observed in our learning times between even and odd values of $N$ are reduced by adding an extra, redundant, neuron in the hidden layer (MLP with an $N/(N + 1)/1$ architecture). The corresponding values of $P(\text{success})$ significantly increase, in particular for $N = 6$, where $P(\text{success})$ increases to 92%. In contrast, for conventional EBP, addition of a single extra neuron in the hidden layer does not change significantly the learning times $^3$).

3.2. Results for the accuracy problem

We present results for example bases with two-dimensional inputs as this allows an easy visualization. Figure 3 schematizes the example bases we have used. The different example bases represent a half-plane (fig. 3a)), a right-angle corner (fig. 3b)) and a rectangle (fig. 3c)). The minus and plus signs represent the locations where the output is required to be negative and positive respectively. The (+) and (-) example classes are separated by strips of width $\Delta X$ where no examples occur. The minimum Euclidean distance between the classes is $\Delta X$ and the 'relative' accuracy required for the separation of these classes is defined as $\epsilon = \Delta X/X_{\text{max}}$. In order to adapt to these example bases, we set our algorithm with the parameters $T_{1(0)} = X_{\text{max}}$, $\rho_{1(0)} = 0.99$, $T_{2(0)} = 1.5$ and $\rho_{2(0)} = 0.9999$. The other parameters were kept as previously defined.

Fig. 3. Example bases to study accuracy-related problems. We represent schematically the example bases we have used for studying situations where a high accuracy is needed in separating example classes. The base in a) defines a half-plane, in b) a right-angle corner and in c) a rectangle. The (-) and (+) symbols represent the locations where the output is required to be negative and positive respectively.
3.2.1. Benchmarking our algorithm for an accuracy problem

Conventional forms of EBP learning proved difficult even for the simplest (half-plane) case. Figure 4 compares learning times for our method with the best results obtained by conventional EBP for the simple half-plane case (five examples in the base). The conventional EBP yields a learning time which diverges roughly as $1/\varepsilon^2$ when $\varepsilon$ decreases. Our method substantially shortens learning times and avoids the divergence when $\varepsilon$ decreases.

3.2.2. The corner and rectangle problems

The corner and rectangle problems prove practically intractable with conventional EBP as soon as the accuracy of separation required, $\varepsilon$, falls below 0.15. This is because accuracy and topological problems are simultaneously involved. In contrast, as can be seen in fig. 5, our method yields reasonable learning times which increase within relatively narrow bounds when $\varepsilon$ is made to decrease. The two-layer MLPs used for these problems had one output neuron and two inputs. As for the hidden layer, it had a single neuron for the half-plane problem, two neurons for the corner problem and six neurons for the rectangle problem. The rectangle problem was also solved with four neurons in the hidden layer (minimal MLP architecture) but the learning times were then doubled. We have checked that the generalization occurs in a 'natural' way, e.g. the output response was negative everywhere in the internal region of the rectangle (fig. 3c)) and positive everywhere in its external region.
3.3. Self-adaptation of the algorithm

In the above, we have seen that the parameters $\eta$, $\zeta$, $\nu$, $\sigma_l$ and $(\beta_1/\beta_2)$ can be set to near-optimal predetermined values. The other adjustable parameters $T_l^{(0)}$, $\rho_l$ and $T_c$ have optima depending on the nature of the example base one deals with. However, one can always set $T_l^{(0)}$ to a value corresponding to the maximum of the inputs from the layer $l - 1$, $\rho_l$ to values near to unity (0.999 say) and $T_c$ to a value well below the smallest estimate for the minimum Euclidean distance between the example classes. The corresponding success probability will then always be high although the learning time may be non-optimal.

3.4. Non-linearity of the sigmoids

In our approach, we control the degree of saturation of the sigmoids in the course of the learning iterations. This is achieved by imposing a unity modulus constraint on the synaptic vector converging on each neuron and by reducing the temperature $T_l$ at the end of each epoch. This proved to be of importance for achieving a good learning performance in MLPs. At the start of the learning iterations, each sigmoid operates near its linear region for the different possible combinations of the corresponding inputs from the previous layer. At the end of the iterations the temperatures should be small enough and the sigmoid response sharp enough to allow effective separation of the different example classes.
Accelerated EBP learning algorithm

3.5. Computational aspects

Our algorithm introduces new computational steps in each iteration as compared with conventional forms of EBP. However, these computations mainly concern neuron-related variables. For a large MLP, by far the most computationally extensive part consists in synaptic updates which are carried out according to a simple local rule (eq. (15)). This major part of the computation has similar complexity to that of a conventional EBP step and can be implemented on parallel neurocomputing hardware\textsuperscript{10}).

The constraint we impose on the modulus of the synaptic vectors keeps the dynamic range of the synaptic coefficients within limited bounds. This makes our algorithm suited for implementation on neurocomputing hardware where the synaptic coefficients are stored as fixed-point variables.

Our algorithm proved to be numerically stable. We observed only a modest improvement when we experimentally introduced a momentum term\textsuperscript{3}). The results presented in this article were all obtained without such a term. Doing away with the momentum term simplifies the hardware requirements and in particular we have no need for extra memories to store the synaptic corrections introduced in the successive learning steps.

4. Summary and conclusions

In this article, we propose a new learning method for MLPs that aims at rationalizing the very popular EBP algorithm. To do so, several new features are introduced, in particular the following.

1. We use a bounded conjugate gradient approach to modify the net inputs to the different neurons. This is performed while minimal perturbation of the neural net is imposed for each learning step.
2. At the start of the learning iterations, we encourage the occurrence of suitable sign patterns for the net inputs of the neurons. For later iteration steps, we gradually request more accuracy.
3. The degree of saturation of the neuron sigmoids is kept under control by imposing a constraint during learning on the amplitudes of the synaptic coefficients converging on each neuron.
4. Different temperatures are attributed to the different neuron layers in order to adapt to the dynamic ranges of the corresponding net inputs. These temperatures are exponentially decreased for successive learning epochs.

In spite of the refinements introduced by our algorithm, its computational complexity for each learning step remains comparable with that of
conventional EBP. In particular, the synaptic coefficients are updated according to a simple local law. This makes it suited for implementation on parallel neurocomputing hardware such as that developed in our laboratory (LEP). Furthermore, it is stable enough to yield good results without the inclusion of a momentum term. There is consequently no need for extra memories to store the corrections introduced during successive learning steps.

A parametric study of our algorithm shows that all the features introduced are significant and bring improvements in the learning performances. The optimal parameter settings are nearly identical for all the cases studied. The learning performance proved to be tolerant to large variations of the algorithm parameters around their optimal values.

We benchmark our algorithm for different problems with respect to more conventional forms of EBP. Two types of learning difficulties are studied, namely topological complexity and high-accuracy requirements to separate example classes properly. For all cases, we reduce learning times by orders of magnitude and we succeed in achieving satisfactory learning for example bases for which conventional EBP failed.

5. Recent results obtained after write-up of this article

After write-up of this article, we have experimented with our algorithm on several real-size applications. In particular, S. Makram-Ebeid and J.-P. Roux from LEP have achieved in collaboration with Christian Wellekens and Hervé Bourlard, from Philips Research Laboratory in Brussels, good learning performances on supervised segmentation of continuous-speech cepstral data. The architecture used was of the Net-Talk type. Each of the continuous sentences used consisted of a series of seven German digits (telephone numbers). The result thus obtained for a test data set was an 86% phoneme frame recognition rate for a male speaker and 83% for a female speaker (27 phoneme classes). We directly used continuously coded cepstral data and needed about 6 to 8 epochs (about 15,000 example frames in each of the training and test sets). In comparison, convergence to meaningful results proved to be practically impossible using classical EBP with this type of continuous data representation.

The reason for our success is that our algorithm (unlike conventional EBP) proved to be well-suited for the separation of ambiguous classes (partly overlapping decision regions for different classes). Detailed assessment of our results and comparison with other techniques will be the subject of further publications.
Acknowledgement

The authors would like to thank C.J. Wellekens of Philips Research Laboratory in Brussels for constructive comments on the manuscript.

REFERENCES


Authors

Sherif Makram-Ebeid: Ph.D. Cambridge University, UK, 1976; Lecturer Electronic Engineering Department, Cairo University, Egypt, 1976–1977; Laboratoires d’Electronique Philips, Limeil-Brévannes, France, 1977–1984; Philips Research Laboratories, Eindhoven, 1984–1987. In his initial work at LEP he was active in semiconductor and device physics. Since 1988 he has been involved at LEP in neural net research and engaged in developing related algorithms, software and applications. The applications which have been of particular interest to him include image processing, currently within the Pygmalion ESPRIT II project.