On-Line Learning with Minimal Degradation in Feedforward Networks

Vincente Ruiz de Angulo and Carme Torras

Abstract—**Dealing with nonstationary processes requires quick adaptation while at the same time avoiding catastrophic forgetting. A neural learning technique that satisfies these requirements, without sacrifying the benefits of distributed representations, is presented. It relies on a formalization of the problem as the minimization of the error over the previously learned input–output (i–o) patterns, subject to the constraint of perfect encoding of the new pattern. Then this constrained optimization problem is transformed into an unconstrained one with hidden-unit activations as variables. This new formulation naturally leads to an algorithm for solving the problem, which we call learning with minimal degradation (LMD). Some experimental comparisons of the performance of LMD with back propagation are provided which, besides showing the advantages of using LMD, reveal the dependence of forgetting on the learning rate in backpropagation. We also explain why overtraining affects forgetting and fault tolerance, which are seen as related problems.**

I. Introduction

LEARNING new patterns quickly without dramatically degrading recall of old patterns is a requirement of adaptive on-line systems.

Algorithms of the backpropagation type are not well suited for applications where learning cannot be confined to an off-line phase. If a very different and representative input–output (i–o) pattern needs to be learned after the training of the main set of patterns has been completed, one gets into trouble. There are two possibilities:

—To train the network with the new pattern isolatedly. This may produce catastrophic forgetting of the old information. Then one has to retrain the network with the old information and the new pattern. Although the performance will recover quicker than learning from scratch, the net will still behave very poorly for a long time.

—To retrain the network directly with an appropriate mixture of the new and the old patterns. The net will not suffer from catastrophic forgetting, but a correct response to the new pattern will be available only after a long time. Moreover, the time to recover the previous level of performance over the learning set increases greatly with the growth of this set.

For succeeding in an application of this kind it is clearly necessary to mitigate forgetting. Up to our knowledge, very few works have tackled this issue. Ratcliff [1] and McCloskey and Cohen [2], after many systematic studies, simply arrive to the conclusion that this problem cannot be satisfactorily solved. French [3] claims that the cause of forgetting is the overlap between the representations of the different patterns and, therefore, he modifies backpropagation so as to produce semi-distributed representations. In these representations, only a few hidden units take the value one, while the majority take the value zero. But there is no guarantee whatsoever that in the process of introducing new i–o patterns with this sort of representations there will be no interference with old patterns, even if they have no ones in common. This type of approach has very serious convergence problems and requires a much larger number of hidden units than straight backpropagation [4]. Reducing the distributedness of the representations has also the very undesirable effect of losing some of the most interesting neural network properties, like generalization (as French himself points out) and damage resistance.

Smieja [5] suggests that reducing the length of the weight vector impinging on some hidden units before introducing a new pattern may be a good heuristic to avoid interference.

Brousse and Smolensky [6] hold that there is no forgetting problem in what they call a "combinatorial environment," because in such an environment there are many virtual memories (error-free novel patterns) that do not interfere with old patterns. But their results can be accounted for by the drastic restriction of the possible i–o patterns that can appear in a combinatorial environment (as Hetherington [7] recognizes) and by the use of autocoder networks. Both facts help generalization, as it actually happens also in some of Hetherington's own experiments regarding the influence of increasing the training set size. We think that results about forgetting obtained with autocoder networks and low-error patterns must not be extrapolated to more general situations.

Krusche [8] has pointed out that the huge receptive field of the weighted-sum units is responsible for interference in neural networks. Units with a limited receptive field are increasingly being used [9]–[11]. Locally tuned units that use radial basis functions (RBF) are the common choice. This can be a valid solution, but an important drawback of RBF units is that they need many more examples than weighted-sum units to generalize well, especially in high-dimensional input spaces. The problem comes from the very local representations formed by this type of units, which can be avoided (for instance by allowing large radia). But it is precisely

Manuscript received January 29, 1993; revised August 21, 1994. This work was supported in part by the Comisíon Interministerial de Ciencia y Tecnología (CICYT) under project TIC 91-0423.

V. Ruiz de Angulo is with the Institute for System Engineering and Informatics, Commission of European Communities, Joint Research Centre TP 361, 21020–Ispra (VA), Italy.

C. Torras is with the Institut de Cibernètica, 08028 Barcelona, Spain.

IEEE Log Number 9409150.

1045–9227/95$04.00 © 1995 IEEE

this locality which allows the prevention of forgetting. The more the receptive fields grow and overlap, the more the interference problem comes back to scene. There exists a trade-off between resistance to interference—local representations and generalization—distributed representations.

Our approach does not require information to be stored in special types of representations. In fact we even try to take advantage of the distributed ones. We simply investigate what can be reasonably done to introduce a new pattern into a previously trained network, while increasing minimally the error in the recall of the previously trained items. An algorithm, which we call LMD for "learning with minimal degradation," is developed to accomplish this task efficiently in a general feedforward net.

The previous work most closely related to ours is that of Park *et al.* [12]. They state the problem in a very similar way to that in Section III, but their resolution method is considerably different, it being based on the gradient reduced method for nonlinear constrained optimization. Their cost function is also different, but in Section VI and the Appendix we show that it can be somehow related to ours. The implications of this relation are explored in an experiment included in Section IX-B.

II. *A Priori* Limitations

We would like to warn the reader that success for any procedure with the same objective as LMD must be necessarily limited. For the different settings in which such a procedure can be applied, we will spell out some existing *a priori* limitations.

When a feedforward network with a fixed number of units has enough capacity to encode a fixed set of patterns, there is a bound on how fast learning can take place, since this problem has been proven to be NP-complete [13], [14]. Therefore, we cannot aim at finding a procedure that in approximately constant time learns a new pattern while leaving the old ones intact, because by iterating this procedure learning could be carried out in time linear in the number of patterns.

Now suppose that the chosen architecture is unable to encode all the patterns perfectly. Let E be the error function over the patterns $1 \cdots n-1$, E' be the error function over the patterns $1 \cdots n$, and E_p be the individual error in pattern p. Applying an ideal procedure to learn pattern n when the network is at the minimum of E, the sole unlikely possibility to arrive at the minimum of E' is that $E_n = 0$ at this minimum. Note that in general the value of E in the minimum of E' will be almost surely higher than the minimum of E. Therefore, if the final aim is to arrive at the minimum of E', introducing perfectly the nth pattern can be worse than doing nothing. As you can see in Fig. 1, it is possible that E' grows when we constrain the net to modify the surface it is producing to pass over a point. The network whose results are displayed in the figure has one input and one bias unit connected to one output unit. The axes in the diagram stand for the i–o coordinates, each point thus representing a pattern. The best approximation the network can do of the old patterns is the continuous line. Learning the new pattern while minimizing the error over the old patterns results in weights giving the dashed line interpolation. Note that this figure presents the worst possible case: a huge number of points that only can be interpolated with a surface having low frequency-high amplitude oscillations, a network with few parameters that is completely unable to fit the surface, and a new pattern far away from the mean of the old patterns.

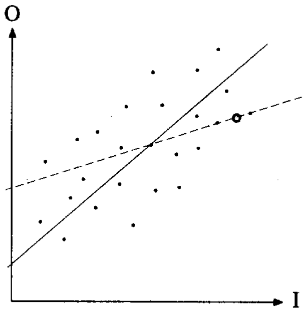


Fig. 1. The points represent the learned patterns, which are approximated by the network with the continuous line. Constraining the network to perfectly encode the new association (little circle), while minimizing the error over the old patterns, gives a global approximation (dashed line) worse than the previous one.

The moral of this discussion is that not only it is impossible to devise a "perfect" algorithm for the nondisruptive encoding of a new pattern, but that even if we had such an algorithm, to apply it indiscriminately in an incremental way could be inappropriate in many situations.

The most natural setting for the application of LMD is that in which the set of patterns to be learned is not fixed but time-varying, in the sense that there is a moving window for the error function so that when new patterns arrive, some of the oldest ones are no longer taken into account. In this case the previous arguments cannot be applied. When introducing a new pattern the emphasis must be put in quick adaptation and some forgetting of the old patterns is desirable. Typical applications of this kind are time series prediction and some control problems.

III. Formalization of the Problem

We can formulate the problem as the minimization of the error over the $n-1$ previously trained patterns, constrained by perfect encoding of the new pattern. It is convenient to consider the current weights (before the application of LMD) as constants and then write each error E_p as a function of weight increments

$$\begin{aligned} \text{Min } E(\Delta W) &= \sum_{p=1\cdots n-1} E_p(\Delta W) \\ E_n(\Delta W) &= 0. \end{aligned}$$

Evaluating the E_p's accurately for a given ΔW would entail presenting the whole set of patterns to the network. If the optimum is to be found through some search process, this evaluation has to be performed repeatedly, leading to a high computational cost.

An alternative solution to accurate evaluation is to approximate the error function over the old patterns through a decoupled quadratic function[1] of ΔW. Then the problem can be written as

$$\min_{\Delta W} F(\Delta W) = \sum_{j,k} c_{jk}\Delta w_{jk}^2 + \sum_{j,k} b_{jk}\Delta w_{jk} \tag{1}$$

subject to the neural network constraints

$$y_j = f_j(x_j) \tag{2}$$

$$x_j = \sum_{k\in \mathrm{Inc}(j)} (w_{jk} + \Delta w_{jk})y_k \tag{3}$$

where w_{jk} is the weight of the connection from unit k to unit j, b_{jk}, and c_{jk} are constants, f_j is the activation function of unit j that given the total input x_j provides its output y_j, and Inc(j) is the index set of the units from which unit j receives direct input. Observe that the y's corresponding to inputs and outputs are constant in this formulation, since they take the values of the new i–o pattern to be encoded, while the y's corresponding to hidden units remain variable.

This problem can be solved through recourse to a standard decomposition technique. Assuming a fixed $Y = [y_j]$, one can find the unique solution $\Delta w_{jk}^*(Y)$ for each decoupled subproblem associated with unit j (j belonging to hidden and output layers) under the assumption that f_j is one-to-one, since each subproblem is a quadratic optimization problem with a linear equality constraint (see the following section). Thus, the initial constrained minimization problem in weight space is converted into an unconstrained one in the smaller space of hidden-unit activations

$$\min_{Y} F(\Delta W^*(Y)). \tag{4}$$

The benefits of this problem transformation are:

—The original constrained minimization problem has been turned into an unconstrained one.

—The new pattern is always perfectly encoded, thus obviating the trade-off between cost minimization and constraint satisfaction mentioned in the preceding section.

—There are far less variables than in the original formulation.

—The domain of each variable is more restricted, because hidden-unit activation functions are normally of limited range.

IV. Resolution of the Subproblem Associated with Each Node

The subproblem associated to unit j can be stated as follows

$$\min F_j(\Delta W) = \sum_{k\in \mathrm{Inc}(j)} c_{jk}\Delta w_{jk}^2 + \sum_{k\in \mathrm{Inc}(j)} b_{jk}\Delta w_{jk} \tag{5}$$

subject to the constraint

$$\sum_{k\in \mathrm{Inc}(j)} (w_{jk} + \Delta w_{jk})y_k - f_j^{-1}(y_j) = 0. \tag{6}$$

This can be easily solved by using, for instance, the Lagrange multipliers method. The function to be minimized becomes

$$G_j(\Delta W) = F_j(\Delta W) + t\left[\sum_{k\in \mathrm{Inc}(j)} (w_{jk} + \Delta w_{jk})y_k - f_j^{-1}(y_j)\right]. \tag{7}$$

Using the fact that $\partial G_j/\partial w_{jk}$ must be zero in the solution, together with the constraint (6), the following expression for the function $\Delta w_{jk}^*(Y)$ can be readily obtained (see [16] for more details)

$$\Delta w_{jk}^*(Y) = \frac{y_k\left(f_j^{-1}(y_j) - \sum_i w_{ji}y_i + \sum_i \frac{b_{ji}y_i}{2c_{ji}}\right)}{c_{jk}\sum_i \frac{y_i^2}{c_{ji}}} - \frac{b_{jk}}{2c_{jk}}. \tag{8}$$

Observe that for the case in which $c_{jk} = 1$ and $b_{jk} = 0$, this is the Widrow–Hoff rule [17]. This formula will be simplified further, but we can conclude now that $\Delta w_{jk}^*(Y)$ is a well-defined function if and only if

a) At least one input arriving to each unit is not zero. This is not a real danger unless threshold units that take zero as one of their states are used.

b) All the c_{jk} are nonzero. We have to take care of this when selecting the parameters of the quadratic function (see Section VI).

c) The denominator of the first fraction must be nonzero. The opposite is an unlikely event that is avoided by choosing positive c_{jk} (see again Section VI).

V. The LMD Algorithm

We can now derive a gradient algorithm for carrying out the minimization in (4). Let us begin by giving $F = \sum_j F_j$ a more explicit form. From (8)

$$\Delta w_{jk}^*(Y) = \frac{\mathcal{E}_j}{c_{jk}\mathcal{M}_j}y_k - \frac{b_{jk}}{2c_{jk}}$$

where $\mathcal{E}_j = f_j^{-1}(y_j) - \sum_i w_{ji}y_i + \sum_i \frac{b_{ji}y_i}{2c_{ji}}$ and $\mathcal{M}_j = \sum_i \frac{y_i^2}{c_{ji}}$. Substituting this expression for Δw_{jk}^* into (5)

$$\begin{aligned}
F_j(Y) &= \sum_i c_{ji}\left(\frac{\mathcal{E}_j}{c_{ji}\mathcal{M}_j}y_i - \frac{b_{ji}}{2c_{ji}}\right)^2 \\
&\quad + \sum_i b_{ji}\left(\frac{\mathcal{E}_j}{c_{ji}\mathcal{M}_j}y_i - \frac{b_{ji}}{2c_{ji}}\right) \\
&= \sum_i c_{ji}\frac{\mathcal{E}_j^2 y_i^2}{c_{ji}^2\mathcal{M}_j^2} + \sum_i c_{ji}\frac{b_{ji}^2}{4c_{ji}^2} - 2\sum_i c_{ji}\frac{\mathcal{E}_j y_i}{c_{ji}\mathcal{M}_j}\frac{b_{ji}}{2c_{ji}} \\
&\quad + \sum_i b_{ji}\frac{\mathcal{E}_j y_i}{c_{ji}\mathcal{M}_j} - \sum_i \frac{b_{ji}^2}{2c_{ji}} \\
&= \sum_i \frac{\mathcal{E}_j^2 y_i^2}{c_{ji}\mathcal{M}_j^2} - \frac{1}{4}\sum_i \frac{b_{ji}^2}{c_{ji}}
\end{aligned}$$

[1] We dispense with cross-terms because, besides adding great complexity to the algorithm, they have costly requirements in computations and memory.

where the definition of $\mathcal{M}_j$ was used in the last step. Observe that $\mathcal{M}_j$ and $\mathcal{E}_j$ can be taken out from the sumatories

$$F_j(Y) = \frac{\mathcal{E}_j^2}{\mathcal{M}_j^2} \sum_i \frac{y_i^2}{c_{ji}} - \frac{1}{4} \sum_i \frac{b_{ji}^2}{c_{ji}} = \frac{\mathcal{E}_j^2}{\mathcal{M}_j} - \frac{1}{4} \sum_i \frac{b_{ji}^2}{c_{ji}}$$

so finally

$$F(Y) = \sum_j F_j(Y) = \sum_j \frac{\mathcal{E}_j^2}{\mathcal{M}_j} - \frac{1}{4} \sum_{j,i} \frac{b_{ji}^2}{c_{ji}}. \tag{9}$$

To greatly reduce complexity, it is convenient to work with the parameters $w'_{jk} = w_{jk} - b_{jk}/2c_{jk}$, since then $\mathcal{E}_j = f_j^{-1}(y_j) - \sum_i w'_{ji} y_i$.

Thus, the first step in the algorithm will be the transformation of all the weights w_{jk} into w'_{jk}, and the last step will be the translation of the optimal hidden-unit configuration into the new increments $\Delta w'_{jk}$, which result also simplified

$$\Delta w'^{*}_{jk} = (w_{jk} + \Delta w^*_{jk}) - w'_{jk} = \Delta w^*_{jk} + \frac{b_{jk}}{2c_{jk}} = \frac{\mathcal{E}_j y_k}{c_{jk} \mathcal{M}_j}. \tag{10}$$

Let us now derive the gradient of F. To prevent that during the search the y's would travel beyond their valid ranges, since the activation functions f_j have usually a limited range, we choose to calculate $\partial F/\partial x_j$. Note that, using $x_j = f_j(x_j)$, x_j appears explicitly in $\mathcal{E}_j$ and therefore in F_j, but also implicitly in all F_s such that $s \in \mathrm{Out}(j)$.

The gradient when j is a hidden unit index is then

$$\frac{\partial F}{\partial x_j} = -2 \frac{\mathcal{E}_j}{\mathcal{M}_j} + 2 f'_j(x_j) \sum_{s \in \mathrm{Out}(j)} \frac{\mathcal{E}_s w'_{sj} \mathcal{M}_s + \mathcal{E}_s^2 y_j / c_{sj}}{\mathcal{M}_s^2}. \tag{11}$$

This formula is valid for networks with whatever number of layers, and even with jumps between layers. The gradient formula is easily implemented by an algorithm with a data flow in two phases, which resembles backpropagation. In the first forward phase, $\mathcal{E}_j$ and $\mathcal{M}_j$ are computed for all units with incoming connections. Then the first gradient term is easily calculated and each of the second term addends is backpropagated to get the total gradient. Do not be misled by the "forward" and "backward" names, since the similarity with backpropagation is limited by the fact that here the information needed by a unit in both phases is already available locally, without any time delay required to wait for information from remote layers. Because of this independence, the updating order can be anyone, allowing even total overlapping. This makes complete parallelism in the implementation possible, not only within a layer, but also among layers.

The LMD algorithm is, therefore, as follows

0) $w'_{jk} \leftarrow w_{jk} - b_{jk}/2c_{jk}$
1) Fix the input and output pattern in the network input and output and derive x_j for all output units. Choose initialization values for all the hidden-unit total inputs x_j.
2) Repeat until a given stopping criterion
    a) Calculate $\mathcal{M}_j$ and $\mathcal{E}_j$ for all units with incoming connections (forward pass).
    b) Backpropagate the second term in (11) and update x_j for all hidden units using $x_j \leftarrow x_j + \mu \frac{\partial F}{\partial x_j}$ (backward pass).
3) Change weights using (10).

Remember that every time x_j is changed, y_j must also be updated because they are binded variables.

It may seem excessive to dedicate a sequence of cycles for only one pattern. Perhaps a solution could be approximated, for example linearizing the network or with another heuristic. It all depends on the difference between a solution of this kind and the true minimum, how this difference is reflected in E, and how the increase in damage has repercussion on the recovery learning time over the previous patterns and the new pattern. It can be supposed that it is worthwhile to spend some more time with only one pattern, trying to trim a bit E, if in exchange one avoids some cycles over the whole learning set.

VI. Choosing the Coefficients of the Cost Function

The most obvious election for the coefficients b_{jk} and c_{jk} is that yielded by an instantaneous second-order approximation of the error function $E(\Delta W)$ ignoring the off-diagonal terms. Then, $c_{jk} = \partial^2 E/\partial w_{jk}^2$ and $b_{jk} = \partial E/\partial w_{jk}$.

With this choice, F is exactly the local estimation of E assumed by several authors [18]–[20] to justify this simple pseudo-Newton rule for optimizing E

$$\Delta w_{jk} = -\mu \frac{\partial E}{\partial w_{jk}} \bigg/ \frac{\partial^2 E}{\partial w_{jk}^2}.$$

Note that step 0) in the LMD algorithm, when the $c_{jk} \neq 0$, is exactly one of these pseudo-Newton steps. The implicit aim of this step is to bring the network to the minimum of F, cancelling out first derivatives. Unfortunately, the minimum of F normally is not the minimum of E, and first derivatives could remain still significative. The conclusion is that this step is scarcely useful and, in fact, what we really need is to minimize the first derivatives of E by whatever means before the aplication of LMD. Later in Section X-B, we will give some advice to reduce first derivatives during training.

Then, if step 0) is taken out of the algorithm, we are minimizing the cost function

$$F(\Delta W) = \sum_{j,k} \frac{\partial^2 E}{\partial w_{jk}^2} \Delta w_{jk}^2.$$

In the minimum, this is still a diagonal second-order approximation. In an arbitrary point, it can be shown [21] that, for a function of the type $\sum_{jk} c_{jk} \Delta w_{jk}^2$, this choice of coefficients is in average the best for estimating E.

To prevent the nullity of the c_{jk} to fulfill condition c) in the last section, we can add a very little constant (range of the hidden activation function/1000, for instance) to everyone of the second derivatives. Even when some c_{jk} are not null, but very close to zero, this helps to ameliorate the behavior of the algorithm. Because, as we said, it is necessary to have positive c_{jk}, we should take absolute values of the second derivatives. Note that when the network is in the minimum, these second derivatives are already positive.

The cost function in [12] is based on approximating $\sum_p \Delta E_p^2$ by means of the network output sensitivities to weight changes. In the Appendix, we show that the sum of the diagonal terms of this cost function can be approximated by a weighted sum of the second derivatives of error components, the weights being the error in the corresponding components. This weighting could be useful in practice in points out of the minimum. In points with almost zero error, however, these coefficicients lose their sense.

There exists another way of using the coefficients to realize a coarser estimation of the damage to the net. By making all the c_{jk} parameters equal to one and all the b_{jk} equal to zero, the algorithm is somewhat simpler and permits either saving the cost of calculating c_{jk} (though it is relatively cheap) or working when they are not available. What LMD is calculating in this case is the nearest solution for the new pattern in the weight space. This is a good heuristic to look for the intersection of the solution space of patterns $1, \cdots, n-1$ and the solution space of pattern n. Under total uncertainty about the shape of the solution space for the new pattern, using this version amounts to introducing white noise in the network with a uniform probability distribution. This type of noise seemed to do little injury in a study performed by Hinton and Sejnowsky [22].

In sum, two versions of LMD have been mainly explored in the experiments: the standard one that uses $c_{jk} = |\partial^2 E/\partial w_{jk}^2|$ and the coarse one that uses $c_{jk} = 1$.

VII. Implementations Details

One of the features that must characterize the application of the algorithm is total automation. We can neither expect to test and correct parameters each time we introduce a new pattern, nor to watch over to decide convergence completion. Besides, we need a reliable algorithm in all situations to bring the network to the minimum without risk of catastrophes. Therefore, in the following subsections we describe the rationale underlying the determination of parameters and initialization values.

A. Advance Rate

The current implementation of LMD follows pure gradient descent but with adaptive step size. The strategy is very simple. When the last step is beneficial (i.e., it leads to a decrement in the cost function), the advance rate μ is multiplied by a number μ^+ slightly greater than one. In the opposite case (increment in F) the step is partially undone, the advance rate reduced by a factor of μ^- and then the step is redone. This can be implemented with little more computation than a forward pass. The advance rate control routine, that runs after step 2b), is as follows

```
if $\Delta F < 0$ then $\mu \leftarrow \mu\mu^+$
while $\Delta F \geq 0$
    $x_j \leftarrow x_j - (1-\mu^-)\mu \frac{\partial F}{\partial x_j}$
    $\mu \leftarrow \mu\mu^-$
    Forward pass 2a)
end while.
```

We have always taken $\mu^- = .5$ and $\mu^+ = 1.3$. Observe that, according to (9), ΔF can be easily calculated from the old value of F and the new values of $\mathcal{E}_j$ and $\mathcal{M}_j$ obtained in the forward pass. The $\partial F/\partial x_j$ are always those computed in 2b) and thus, they do not need to be recomputed. The initial value of μ is only important for a quicker convergence, because convergence is guaranteed. The most convenient value depends on the size of the network and, therefore, for the scaling experiment we scale also the initial μ; in the other experiments the networks are of comparable size and a value of two is always used. As a refinement to the basic algorithm, it is possible to use a much larger μ^+ and a very small μ^- in the first iteration of LMD until a mistaken step is done (if the initial step is not overshot) or a valid step is done (if the initial step is overshot) and afterward use the normal values.

B. Stopping Criterion

The search is finished when

$$\sqrt{\frac{\sum_{j/\mathrm{Inc}(j)\neq\emptyset}\left(\frac{\partial F}{\partial x_j}\right)^2}{n_H}} < G_{\min}$$

where n_H is the total number of hidden units and $G_{\min}$ is a constant that regulates the accuracy in finding the minimum. Dividing by $\sqrt{n_H}$ seems convenient to obtain values independent of the network dimensionality. In normal practice, $G_{\min} = 0.005$ seems a good choice, and this is the value used in all the experiments reported.

C. Initial Hidden-Unit Configuration

We have used several architectures with different weights to test the existence of local minima. This was done by using a great number of random initial hidden-unit configurations and measuring the cost function in the final points reached. The result has been that networks that are able to learn, i.e., those with moderately saturated units, rarely lead to landscapes with local minima. Only using random weights of increasing magnitude, local minima begin to appear more numerous and higher. Because of the scarcity of local minima, the initial hidden-unit configuration is not a crucial question, but the number of cycles required for convergence can increase with a bad selection of the initial point.

There seem to be two privileged initial points: one is the hidden-unit configuration that results from propagating activity through the network and the other is the one with all the unit activations in the middle of the activation range. The first is good in the case that the error in the new pattern is low, since then the weight modifications needed to get the new pattern are small, and as a consequence the ideal hidden-unit configuration is near this point. The second point has the advantage that each hidden unit is completely free to go in one or other direction (in fact, this is also useful to avoid local minima) and the gradient of the activation function is maximum (at least for sigmoids). Here all the experiments use the second option. A more elaborate decision could be to switch to the first option when the error in the new pattern is small.

VIII. Performance Measures

In this section we develop tools for evaluating the computational savings provided by LMD, which turn out to have wider application.

One method of measuring the benefits of using LMD would be comparing E', the global error over the patterns $1 \cdots n$, before and after applying LMD to the nth pattern. This should be indicative of how much the search of the E' minimum is facilitated. Surprisingly, it is not like this. For instance, if the error after the application of LMD is slightly higher, we have systematically observed that LMD helps nevertheless to shorten learning times. Probably this phenomenon is similar to the accelerated relearning times registered in [22] when some disturbance is introduced in trained networks.

It is a knotty problem to measure the computation costs of arriving at a minimum from two different points. Here we suggest two measures that are independent of any algorithm parameters and rely only on the landscape of the error function. Therefore they reflect objectively some aspect of the difficulty to find a minimum from different initial points. We have checked that both give results qualitatively similar if it is not required to arrive very accurately to the minimum. For instance, the phenomenon mentioned above appears independently of the measure used.

The first measure is the time a dynamical system driven by the system of equations

$$\frac{\partial E}{\partial W} = \frac{\partial W}{\partial t}$$

would take to go from one point to another of its trajectory. We call this measure backpropagation time, because these are the learning equations of backpropagation as a continuous dynamical system. To estimate backpropagation time we constrain pure gradient backpropagation to take only steps which produce error decrements that can be predicted by a linear approximation of the error function, i.e., we take the criterion

$$\frac{|\mathrm{Est}\Delta E^{k)} - (E^{k+1)} - E^{k)})|}{\sqrt{(E^{k+1)} - E^{k)})\mathrm{Est}\Delta E^{k)}}} < \text{exigence} \qquad (12)$$

to accept the learning rate $\mu^{k)}$ that produced $\Delta w_{jk}^{k)}$ as correct, where $\mathrm{Est}\Delta E^{k)}$ is the linear estimation of the error increment based on the first derivatives

$$\mathrm{Est}\Delta E^{k)} = \sum_{j,i} \Delta w_{ji}^{k)} \frac{\partial E^{k)}}{\partial w_{ji}}.$$

If all the steps along the learning process satisfy this criterion, we can say that the trajectory followed by the algorithm approximates the trajectory that would follow the dynamical system above. The fidelity to the continuous path will be controlled by the exigence parameter.

If step k satisfies the criterion, $W(t)$ is approximately linear in the section between $W^{k)}$ and $W^{k+1)}$, and, thus, we can estimate the time to perform $\Delta W^{k)}$ by dividing directly the distance by the instantaneous velocity of the system, the gradient norm

$$t(W^{k)}, W^{k+1)}) = \frac{\|\Delta W^{k)}\|}{\|\frac{\partial E}{\partial W}\|} = \mu^{k)}.$$

Therefore, the time to complete a trajectory is $\sum_k \mu^{k)}$ when all the steps satisfy the linear constraint (12).

It is possible to adapt μ near optimally during the training with an algorithm similar to the one presented below.

Using this measure, we have observed that the backpropagation time required to eliminate the last residuals of the error is much bigger than the that needed to eliminate the main part of the error. This is due to the fact that velocity slows down very much in flat regions and especially in the neighborhood of a minimum. Algorithms more sophisticated than raw backpropagation are expected to behave in a rather different manner.

We propose another measure that overcomes the above shortcoming, while at the same time allowing quicker computation. The measure is the standard curve length defined for rectificable functions

$$\wedge(t_1, t_2) = \int_{t_1}^{t_2} \left|\frac{\partial E}{\partial W}\right|(t)dt$$

where t_1 is the initial time point and t_2 is the final time point. We could compute it in a similar way to the one used for back-propagation time, taking only linearly predictable steps. Instead, we have developed a more efficient, although more complicated algorithm, that will be of use later for other purposes.

The idea is that to calculate curve length approximating system trajectory we can relax the linearity constraint of magnitude predictability to only angle continuity between two steps, i.e., we only accept one step if the angle with the next step is close enough to zero. In this way we profit from the fact that, unlike back-propagation time, curve length is independent of the velocity with which E comes down and can always be computed in one step in the zones where the gradient direction does not change. The only inconvenience, which adds some complexity to the algorithm, is that to know whether one step is too big to be accepted, we must know the direction of the next step. This is the algorithm used to estimate curve-length

```
repeat until stopping criterion
    $\Delta W \leftarrow \mu\partial E/\partial W$
    $W \leftarrow W + \Delta W$
    while ang$(\Delta W,\ \partial E/\partial W(W)) >$ exigence
        $W \leftarrow W - \alpha\Delta W$
        $\Delta W \leftarrow (1-\alpha)\Delta W$
        $\mu \leftarrow (1-\alpha)\mu$
    $\Lambda \leftarrow \Lambda + \|\Delta W\|$
$\mu \leftarrow \beta\mu$.
```

Exigence regulates the fidelity to the continuous system trajectory. m is adapted to grow slightly at a geometrical rate of $\beta > 1$ when one step is accepted and decrease more quickly at a rate of $1 - \alpha$, $1 < \alpha < 0$ when it is not. This is to keep m near the highest values allowed by the angle continuity constraint. In our simulations, $\alpha = 0.5$ and

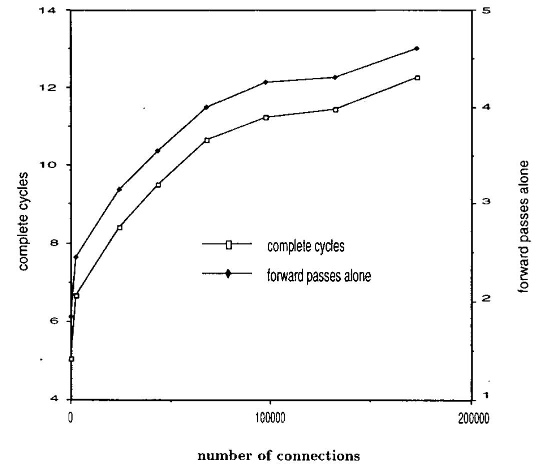


Fig. 2. Scaling experiment: Number of LMD complete cycles and feed-forward steps alone (due to overshot steps) as a function of network size (number of connections).

TABLE I
ERROR INCREMENTS AFTER APPLYING LMD WITH $c_{jk} = 1$ AND $c_{jk} = |\partial^2 E/\partial w_{jk}^2|$. THE EXPERIMENT WAS PERFORMED WITH DIFFERENT NUMBERS OF ALREADY LEARNED PATTERNS. THE RESULTS ARE AVERAGED OVER 20 NEW PATTERNS

Architecture	8-3-1		13-12-4		300-30-10	
Nr. of Patterns	5	10	10	25	50	150
Coarse LMD	0.064	0.111	0.018	0.032	0.002	0.002
Standard LMD	0.38	0.087	0.012	0.028	0.001	0.002

$\beta = 1.2$. To compute $\text{ang}(\Delta W, \partial E/\partial W(W))$, a complete back-propagation cycle is required to get the E gradients (which, when the while condition fails, can be reused without further computations for the next step), but weights and weight increments must not be updated.

IX. Experimental Results

A. Scaling Properties

Fig. 2 gives an idea of the number of steps needed to reach the minimum and the scaling properties of LMD. Networks of several sizes were used in this experiment, all with the same proportion of units in each layer. The smallest was a 8-3-1 network and the others were obtained by multiplying by 10, $20\cdots$ the number of units in each layer of this network. The initial μ for the 8-3-1 network was one, and was multiplied in the same manner above for the rest of the networks. The abcises indicate the number of connections. The ordinates show the average of forward–backward cycles and forward steps alone (due to mistaken steps) for 20 random new patterns. The cost function was $F = \sum \Delta w_{jk}^2$ and each weight w_{jk} in the network was obtained randomly with the uniform distribution $[-1.5/|\text{Inc}(j)|, 1.5/|\text{Inc}(j)|]$.

It can be seen that the number of steps is not excessive, indicating the great simplicity of the unconstrained search space. Scaling properties are good, making less indispensable the development of an acceleration algorithm. Other experiments also indicate that scaling with the number of layers is better than for backpropagation.

B. Solution Quality for Different Coefficients Settings

Table I presents results for three different networks trained with two different numbers of patterns. It shows the average error increment in the output units for the previously trained pattern when LMD is applied, with $c_{jk} = 1$ and $c_{jk} = |\partial^2 E/\partial w_{jk}^2|$. The error measure used is

$$E = \frac{1}{2}\frac{1}{n\text{No}} \sum_{\substack{j=1\cdots n \\ i=1\cdots \text{No}}} \varepsilon_{ji}^2$$

where No is the number of output units and ε_{ji} is the difference between the desired value and the network response value in ouput unit i for pattern j. Net structure is expressed in the obvious way (net 8-4-1 has eight input units, four hidden units, and one output unit). Both the previously learned patterns and the new ones were generated randomly from a uniform distribution in $[-1.2, 1.2]$. A symmetric sigmoid ranging from -1.712 to 1.712 was used for the hidden-unit activations. The activation functions of the output units were linear. Take into account that these conditions are bad to reduce the error increment: Linear output units allow the error to grow unlimitedly, and the big ranges of the i–o patterns and hidden-unit activation functions lead to a great variance in the output of the network. The table shows averages over 100 new patterns. When the networks have been trained with a lot of patterns, the error increments are larger for the two versions. This is due to the fact that a network with more random patterns has more output variance, and thus the error average of the new learned random patterns is higher (more than double in the network 32-12-4). Also the advantage of the standard version over the coarsest one is lower with more patterns, because second derivatives are more uniform, indicating that the network is more saturated with information, parameters are less free to vary and, as a consequence, there are no privileged directions. When the patterns are not random, the networks become saturated more gradually. We can also guess that, in the case of many patterns, the quadratic estimation of the error function is poorer, because the very large errors force the network to look for solutions far from the present position in weight space.

Finally, another experiment was made to test the solution quality provided by differet settings of the coefficients in networks outside the vicinity of a minimum of the learning set. Table II presents results for the 8-3-1 architecture trained with five patterns until some fixed error is reached. Five networks with the error levels shown in the table were used, and each of them underwent the introduction of a new pattern with the coarse and standard versions of LMD. Also the weigthed squared derivatives (15) inspired in [12] and derived in the Appendix were tested for comparison. As before, results were averaged over 20 new patterns. It can be noticed that the standard version of LMD still gets significatively better results than the coarse one. On the other hand, the weighted squared derivatives (15) work also well, but when the error over the old patterns is very low they become very risky.

TABLE II
EXPERIMENT OUT OF THE MINIMUM. ERROR INCREMENTS ARE SHOWN FOR THE TWO DIFFERENT VERSIONS OF LMD IN A NETWORK WITH DIFFERENT ERROR LEVELS IN THE PREVIOUSLY LEARNED PATTERNS. AN 8-3-1 ARCHITECTURE WITH FIVE OLD PATTERNS WAS USED. THE FIGURES REPRESENT AVERAGES OVER 20 NEW PATTERNS. THE RESULTS FOR LMD ARE COMPARED WITH USING THE C_{JK}'S DEFINED IN (15), WHICH ARE LABELED AS WEIGHTED SQUARED DERIVATIVES

Old patterns Error	.1	.05	.01	.001	.0005	.0001
Coarse LMD	.049	.054	.057	.061	.062	.063
Standard LMD	.023	.025	.029	.033	.034	.035
Weighted Squared Derivatives	.030	.034	.042	.083	.121	.235

TABLE III
THE FIRST ROW SHOWS THE CURVE LENGTH FROM THE POINT OF MINIMUM ERROR FOR THE SET OF PATTERNS $1, \cdots, n-1$ TO THE VICINITY OF THE POINT OF MINIMUM ERROR FOR THE SET OF PATTERNS $1, \cdots, n$. IN THE SECOND ROW, THE MEASUREMENTS WERE MADE TAKING AS INITIAL POINT THAT RESULTING FROM APPLYING LMD TO THE nTH PATTERN. AVERAGES OVER 20 NEW PATTERNS ARE SHOWN

Architecture	8 - 3 -1		32 - 12 - 4		300 - 30 - 10	
Nr. of Patterns	5	10	10	25	50	150
Length (without LMD)	0.335	0.546	0.334	0.456	0.182	0.169
Length (After LMD)	0.088	0.397	0.059	0.262	0.002	0.061

C. Computational Savings Derived from the Application of LMD

We present now some experimental results regarding the improvement in training times provided by the use of LMD. The networks are the same as those in the preceding experiment, with the same already learned patterns and the same 20 new patterns. Let $W^*_{1 \cdots n-1}$ be the weight configuration at which the minimum error $E^*_{1 \cdots n-1}$ for the set of patterns $1 \cdots n-1$ is attained, and E be the error function over patterns $1 \cdots n$. For each network and new pattern, we first calculate very accurately the minimum error E^* for the set of patterns $1 \cdots n$, then we measure curve length from $W^*_{1 \cdots n-1}$ to the first point in the trajectory with an error minor than $E^* + .2(E(W^*_{1 \cdots n-1}) - E^*)$, and after from the same original point transformed by applying LMD with pattern n, to the first point with the same level of error as before. Table III shows the results. The differences in curve length should be indicative of the advantage of using LMD to tune the network before applying any algorithm of the backpropagation type, because length only depends on the error function shape. The computational effort to introduce the new pattern with LMD is negligible, since the number of LMD cycles is usually less than the number of patterns in the training set. The table reveals that it is more advantageous to use LMD when there are few patterns than when there are many. The main reason is the same one pointed out in the preceding subsection: growth of the new pattern errors leads to growth of the damage to the network. This is a very abnormal situation, however, because in typical applications the error in the new patterns tends to decrease when the number of learned associations grows. Moreover, to be always advantageous under this incremental scheme, LMD (and any other algorithm) needs to be presented with new patterns yielding progressively smaller errors, because in the long term a highly-erroneous new pattern can lead to a situation of the type we talked about in Section II and exemplified in Fig. 1. This shortcoming could be overcome, for example, by reducing wisely only a fraction instead of the complete error of the new pattern. We have left these refinements for future work.

D. LMD Versus Backpropagation

We present now some experiments comparing LMD and standard backpropagation with different advance rates μ. This parameter turns out to be a very important factor in the comparison.

In Fig. 3(a) the net 32-12-4, trained with 10 random patterns, undergoes the introduction of a new pattern with different backpropagation learning rates. The pattern is considered to have been learned when the average absolute error in the output units is 0.01. The distance from the starting point to the final one in the weight space and the number of cycles needed are shown. The limit to which the distance tends when $\mu \rightarrow 0$ can only be approximated with large computational costs. Fig. 3(b) displays how the distances to the starting point showed in Fig. 3(a) affect the error in the old patterns and how this in turn affects the recovery time (measured in terms of backpropagation time) needed to relearn both the new and the old patterns. The distance lower bound for backpropagation when $\mu \rightarrow 0$ (0.69), as well as results with the coarsest ($c_{jk} = 1$) and standard ($c_{jk} = |\partial^2 E / \partial w^2_{jk}|$) versions of LMD are also shown here.

The different weight solutions provided by back-propagation are in random directions with respect to the starting point, because they were obtained independently of E, but a neat linear relation appears between the distance and the error increment. An important finding for our investigation is that the recovery time follows an exponential-like curve, implying that it is very important, with respect to recovery time, to trim as much as possible the damage caused to the old patterns, even if the possible reductions are small.

This example also shows how the distance for the coarsest version of LMD must be by definition of the cost function always the shortest one. The difference between the true minimum distance found by the coarsest version of LMD and the approximation made in the limit by back-propagation is variable, especially for highly erroneous new patterns, but usually the two points are close.

Finally, note that the standard version of LMD provided by far the best results (very small error increment and recovery time), and it did so by moving a long distance away from the initial point, thus proving the existence of a privileged direction.

X. DISCUSSION

A. The Influence of the Backpropagation Advance Rate on Forgetting

One of the main results we have obtained is the dependence of backpropagation forgetting on the learning rate. Cohen [2]

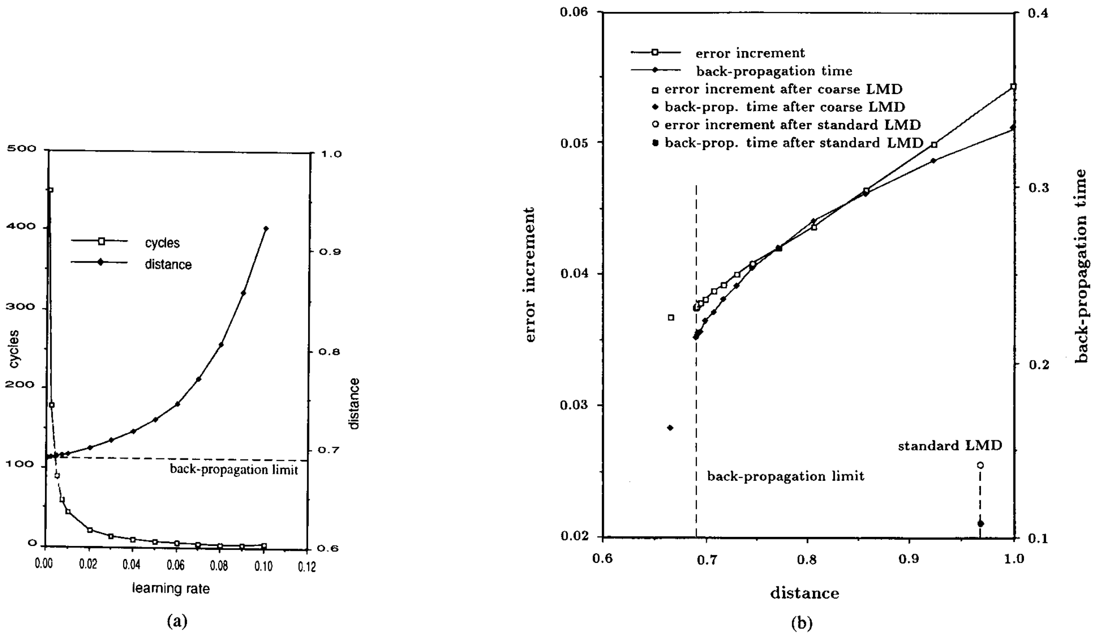


Fig. 3. (a) Starting with a network in the minimum of the error function for the old patterns, a new pattern is introduced with different back-propagation learning rates. The graphic shows the number of cycles needed to learn the pattern and the total distance covered from the initial point in each case. It is clear that, when the learning rate tends to zero, the distance tends to a limit. (b) Another aspect of the experiment described in (a). The distances represented in (a) are now in the axis of abcises. The two curves represent the error increments and the backpropagation times beginning after the introduction of the new pattern and ending in the global minimum of the error function including the new pattern. Coarsest LMD ($c_{jk} = 1$) minimizes explicitly distance and so, it must obtain results better than the backpropagation limit. Standard LMD ($c_{jk} = |\partial^2 E/\partial w_{jk}^2|$), however, which does not take into account distance, gets the best results.

already observed this in a systematic variation of all backpropagation parameters. We can now provide an explanation of this dependence. The gradient of the error function for the new pattern can lead the network anywhere in principle, but it is a good heuristic for finding one of the nearest solutions. The problem is that backpropagation with usual learning rates does not follow the true gradient line because of its discrete nature. Since the solutions for the new pattern pervade the weight space, a too big step may lead to a point crossed by a gradient line driving the network to a different and farther solution. If backpropagation is forced to closely follow the true gradient descent line (as it is the case when the advance rate tends to zero), it becomes a reasonable application of the coarsest version of LMD, but with high computational cost. Usual accelerating algorithms, taking bold steps, can only worsen forgetting.

Contrarily, the algorithm for measuring curve length follows the gradient line with the desired accuracy, but with the highest learning rates possible in each step, alleviating the inefficiency-catastrophic forgetting trade-off in backpropagation, thus finding another use complementary to the one for which it was designed in Section VIII. For instance, applying the curve length algorithm with exigence = .99 in the last experiment of the preceding section, the distance obtained was .693, which is only slightly higher than the backpropagation limit, and only 27 cycles were used, almost half of those needed by backpropagation with the appropriate μ to get the same distance. The algorithm to compute backpropagation time could also do the job in a simpler but more inefficient way.

B. How to Prepare a Network for Damage or the Relation of LMD with Fault Tolerance

A conclusion from the exponential-like curve for the recovery time reported in Section IX-D, as well as the reasoning about the convenience of dispensing with the b_{jk} parameters presented in Section VI, is that one must wait as long as possible to the completion of the learning of the previous patterns (by second-order, standard backpropagation or whatever means) before introducing the new patterns.

The overtraining effect had been observed, but not explained, in studies of forgetting [2] and fault-tolerance [23]. Avoidance of forgetting and increasing fault-tolerance can be seen as intimately related goals. Both try to minimize the effect of weight perturbations on the information stored in the network. The only difference is that, to avoid forgetting, one can control somewhat the form of the perturbation. This similarity can be profited directly. Here, for example, the explanation for damage reduction after overtraining can be easily transferred from one domain to the other.

Through the Taylor series expansion of the error-increment function produced by a perturbation, it is evident that minimizing the first derivatives of E (the b_{jk} parameters) is the first priority to reduce the error increment. A minimum of E is also a minimum of the absolute value of its first derivatives, but moving across the weight space while decreasing the error function does not imply decreasing derivatives, unless the network is really near the minimum of E. Examining the curves produced by the learning rate = .002 in Fig. 4(a)-(b), it can be seen that, when the training is finished, the

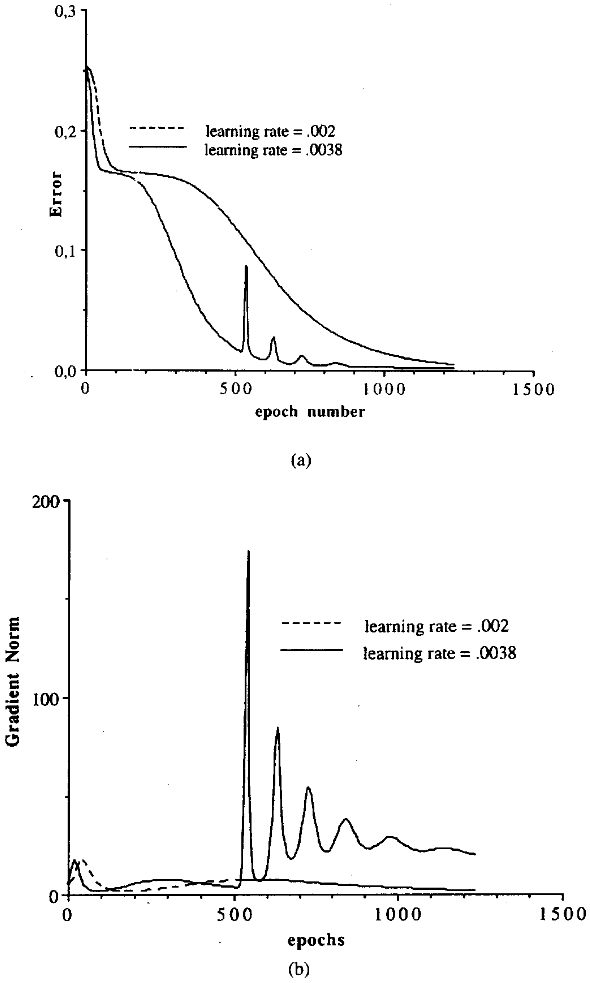


Fig. 4. (a) Error evolution in a network with two different learning rates. (b) Gradient norm evolutions along the learning corresponding to the training curves in (a).

level of error is very good, practically zero, but the gradient norm is still relevant, of the same order of that found in the middle of learning. This is the effect of the velocity difference in minimizing E and its derivatives. With enough training (overtraining) both values can be brought to zero.

On the other hand, with a higher learning rate of .0038, the learning curve fluctuates but arrives faster to the minimum. Notice that, in the last part of the training, the curve stabilizes and descends uniformly, arriving at a level three times lower than before, but nevertheless the final gradient norm is huge. Even a minimal modification of the weights will produce catastrophic forgetting. The different versions of backpropagation are normally used with the highest learning rates that allow faster training in the long term. As a consequence, in many occasions (even if the error function is always decreasing) the network is often out of the bottom of the error function valleys, in points with high derivatives. Then if one wants to alleviate the perturbation effects in a given stadium of the learning, the best one can do is to minimize locally the derivatives of E following for a certain time the true gradient of E. This will bring the network to the bottom of the current valley. To follow the gradient line, one can make some steps of backpropagation with a very cautious learning rate or, more efficient and safe, one can use one of the algorithms presented in Section VIII, which find thus here still another use.

C. The Relation of LMD with Pruning

The standard version of LMD, without b_{jk} and with c_{jk} as second derivatives in the minimum of E, minimizes the same function (constrained by the new pattern) as Le Cun *et al.* [24] in their pruning procedure, our c_{jk} being their sensitivities. In a certain sense, our technique can be seen as opposed to pruning. Pruning detects the less profited weights to eliminate them. Instead, LMD uses them to introduce new information. The relation with pruning suggests that advances in pruning techniques can be incorporated into LMD. For example, some authors have recently used weight sensitivities that go beyond the strict locality of second derivatives [25].

XI. Conclusion

After pointing out the theoretical reasons that prevent complete success in avoiding forgetting in neural networks with distributed representations, we have developed a theoretical framework for the problem which leads naturally to the LMD algorithm as the more efficient way to tackle it. Full parallelism, both within and between layers, is one of the features of LMD. We have shown results with the coarse and standard versions of the algorithm, which demonstrate its good scaling properties, as well as the solution quality and computational savings derived from its application. LMD, like the algorithm in [12], can control the comparative importance of the previous data by weighting each pattern through the coefficients c_{jk} in the error function. This feature can be useful in applications of the moving-window type, strengthening for example the remembrance of the last presented patterns. Furthermore, if the coefficients are not rigidly used to estimate the error function, LMD allows a great flexibility in controlling how much of a pattern is learned by each individual connection, each unit or each layer.

Two measures and algorithms for measuring the costs of going from one point to another of the weight space have been developed. These algorithms have characteristics that make them interesting for some other purposes, some of which have been pointed out. They could be interesting also for those who want to evaluate the goodness of some initial weights for faster learning [26].

The relationship between avoiding forgetting, fault-tolerance, and pruning has been shown, and some advice has been given for increasing resistance to damage in a network. Besides, we have shown that when the learning rate of back-propagation tends to zero, the solution found approximately minimizes $\sum \Delta w_{jk}^2$, which gives a crude second-order approximation of the error increment over the previously learned patterns when the network is at a minimum.

We have also shown that the complete avoidance of catastrophic forgetting has, however, *a priori* limitations for standard back-propagation networks with fixed size. Future work will address the development of a wider framework for networks including both sigmoid and RBF units. For the sake of clarity, we derived LMD on the basis of hidden-unit activations, but it can be based as well on the total input to the hidden units, so that the assumption of invertible activation functions would no longer be needed. Also the dot product can be substituted by the Euclidian distance, thus allowing the use of the typical Gaussian units.

It is also worthwhile to design a general learning algorithm with LMD at its base. This algorithm will profit from the direct manipulation of the internal representations, just as other similar algorithms presented recently do [27]–[29]. Another minor question is the investigation of acceleration techniques for LMD.

Appendix

In [12], the effect of ΔW on the previous data is modeled as

$$\sum_p \Delta E_p^2$$

which is in turn approximated by taking linear estimations of each of the E_p

$$F(\Delta W) = \sum_{p=1\cdots n-1} \left(\sum_{j,k} \frac{\partial E_p}{\partial w_{jk}} \Delta w_{jk} \right)^2. \qquad (13)$$

Developing $\partial E_p / \partial w_{jk}$

$$F(\Delta W) = \sum_{p=1\cdots n-1} \left(\sum_{j,k} \sum_r \frac{\partial O_r(p)}{\partial w_{jk}} \varepsilon_r(p) \Delta w_{jk} \right)^2$$

where $O_r(p)$ is the r output component of the net when the pattern p is presented, $\varepsilon_r(p) = (O_r(p) - O_r^*(p))$, and $O_r^*(p)$ are the desired values. This function can be expressed as

$$F(\Delta W) = \sum_{p=1\cdots n-1} \sum_{j,k,m,n} \sum_{r,s} \frac{\partial O_r(p)}{\partial w_{jk}} \frac{\partial O_s(p)}{\partial w_{mn}} \times \varepsilon_r(p) \varepsilon_s(p) \Delta w_{jk} \Delta w_{mn}.$$

Taking only the diagonal terms j, $k = m$, n and $r = s$

$$F(\Delta W) = \sum_{p=1\cdots n-1} \sum_{j,k} \sum_r \left(\frac{\partial O_r(p)}{\partial w_{jk}} \right)^2 \varepsilon_r^2(p) \Delta w_{jk}^2 \qquad (14)$$

which is a function of the type $\sum_{j,k} c_{jk} \Delta w_{jk}^2$ chosen in this paper, whose coefficients are

$$c_{jk} = \sum_{p=1\cdots n-1} \sum_r \left(\frac{\partial O_r(p)}{\partial w_{jk}} \right)^2 \varepsilon_r^2(p). \qquad (15)$$

It is easy to see $\sum_{p=1\cdots n-1} \sum_r \left(\frac{\partial O_r(p)}{\partial w_{jk}} \right)^2$ is an approximation of $\partial E^2 / \partial w_{jk}^2$, concretely a Levemberg–Marquardt approximation. Thus the c_{jk} coefficients in (15) can be considered as a modified second derivative of E, weighted by the error of the outputs of the network.

Acknowledgment

The authors like to thank an anonymous referee for suggesting the compact problem statement presented in Section III.

References

[1] R. Ratcliff, "Connectionist models of recognition memory Constraints imposed by learning and forgetting functions," *Psychological Review*, vol. 97, no. 2, pp. 235–308, 1990.

[2] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *The Psychology of Learning and Motivation*, G. H. Bower Ed. New York: Academic, 1989.

[3] R. M. French, "Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks," Center for Research on Concepts and Cognition, Indiana Univ., CRCC Tech. Rep. 51-1991, 1991.

[4] J. M. J. Murre, "Categorization and learning in neural networks," Ph.D. dissertation, Univ. Leiden, 1991.

[5] F. J. Smieja, "Hyperplane 'spin' dynamics, network plasticity and back-propagation learning," Tech. Rep. German National Res. Centre for Comput. Sci. (GMD), Nov. 1991.

[6] O. P. Brouse and P. Smolensky, "Virtual memories and massive generalization in connectionist combinatorial learning," in *Proc. 11th Annual Conf. Cognitive Science Society*, 1989, pp. 380–387.

[7] P. A. Hetherington, "The sequential learning problem in connectionist networks," Master's thesis, Dept. Psych., McGill Univ., Montreal, Quebec, Canada, Nov. 1990.

[8] J. K. Kruschke, Message to the Connectionists Electronic Mail Distribution List.

[9] J. Moody and C. Darken, "Fast learning in networks of localy tuned processing units," *Neural Computation*, vol. 1, no. 2, pp. 281–294, 1989.

[10] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE*, vol. 78, no. 9, pp. 1481–1497, 1990.

[11] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, vol. 3, no. 2, 1989.

[12] D. C. Park, M. A. El-Sharkawi, and R. J. Marks II, "An adaptively trained neural network," *IEEE Trans. Neural Networks*, vol. 2, no. 3, May 1991.

[13] A. Blum and R. L. Rivest, "Training a 3-node neural network is NP-complete," in *Proc. 1988 Workshop Computational Learning Theory*, 1988, pp. 9–18.

[14] S. Judd, *Neural Network Design and the Complexity of Learning*. Cambridge, MA: MIT Press, 1990.

[15] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," *Advances in Neural Information Processing I*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kauffmann, 1989, pp. 107–115.

[16] V. Ruiz de Angulo and C. Torras, "The MDL algorithm," in *Proc. IWANN '91*, A. Prieto, Ed., *Lecture Notes on Computer Science* no. 540. New York: Springer-Verlag, 1991, pp. 162–172.

[17] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptative pattern recognition," *Computer*, Mar. 1988.

[18] S. Becker and Y. Le Cun, "Improving the convergence of back-propagation learning with second order methods," in *Proc. 1988 Connectionist Models Summer School*, D. Touretzky, F. Hinton and T. Sejnowski, Eds., 1988, pp. 29–37.

[19] R. Scalatter and A. Zee, "Emergence of grandmother memory in feedforward networks. Learning with noise and forgetfulness," in *Connectionists Models and their Implications, Readings from Cognitive Science*, D. Waltz and J. H. Feldman, Eds. Norwood, NJ: Ablex, 1988, pp. 309–323.

[20] L. P. Ricotti, S. Ragazzini, and G. Martinelli, "Learning word stress in a suboptimal second order back-propagation neural network," in *Proc. IEEE Int. Conf. Neural Networks*, San Diego, vol. I, pp. 355–364.

[21] V. Ruiz de Angulo, Ph.D. dissertation, in preparation.

[22] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzman machines," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol 1: Foundations*, D. E. Rumelhart and J. L. McCLelland, Eds. Cambridge, MA: MIT Press, 1986.

[23] J. Mijhuis, Hofflinger, Van Sc. Haik, and L. Spaanenburg, "Limits to the fault-tolerance of a feedforward neural network with learning," in *Proc. 20th International Symp. Fault Tolerance Computing*, Newcastle upon Tyne, June 1990, pp. 228–235.

[24] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," *Advances in Neural Information Processing Systems*, David S. Touretzky, Ed. San Mateo, CA: Morgan-Kauffman, 1990.

[25] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 2, June 1990.

[26] S. Gavin, "Designing multilayer perceptrons from nearest-neighbor systems," *IEEE Trans. Neural Networks*, vol. 3, no. 2, Mar. 1992.

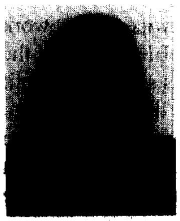
[27] T. Grossman, R. Meir, and E. Domany, "Learning by internal representations," *Complex Syst.* 2, pp. 555–575, 1988.

[28] A. Krogh, C. J. Thorbergsson, and J. A. Hertz, "A cost function for internal representation," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, Ed. San Mateo, CA: Morgan-Kauffman, 1990.

[29] R. Rohwer, "The moving target training algorithm," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, Ed. San Mateo, CA: Morgan-Kauffman, 1990.

Vicente Ruiz de Angulo was born in Miranda de Ebro, Burgos, Spain. He received the B.Sc. degree in computer science from the Universidad del País Vasco.

During the academic year 1988–89, he was Assistant Professor at the Universitat Politècnica de Catalunya. In 1990 he joined the Neural Network Laboratory of the Joint Research Centre that the Commission of the European Communities has in Ispra (Italy). His interests in neural networks include fault tolerance, noisy and missing data processing and their application to temporal series prediction and robot path finding.



Carme Torras was born in Barcelona in 1956. She received the M.Sc. degree in mathematics from the Universitat de Barcelona in 1978, the M.Sc. degree in computer science from the University of Massachusetts at Amherst in 1981, and the Ph.D. degree in Computer Science from the Universitat Politècnica de Catalunya in 1984.

Since 1981 she has been with the Institut de Cibernètica in Barcelona, conducting research on Neurocomputing and Robot Motion Planning. She has been involved in several ESPRIT projects financed by the Commission of the European Communities, among them those entitled "Robot Control Based on Neural Network Systems" (CONNY), "Self-Organization and Analogical Modeling Using Subsymbolic Computing" (SUBSYM), and "Behavioral Learning: Sensing and Acting" (B-LEARN II). At present, she holds a position of Professor of Research in the Consejo Superior de Investigaciones Científicas (CSIC) and she teaches Ph.D. courses in the fields of robotics and artificial intelligence at the Universitat Politècnica de Catalunya.

Based on her thesis, Dr. Torras authored the book *Temporal-Pattern Learning in Neural Models* (Lecture Notes in Biomathematics, Springer-Verlag, 1985).