Learning of Nonstationary Processes

Bad formatted version of the chapter of the book Optimization Techniques , Academic Press

Index

Neural Network Learning of Nonstationary Processes

V. Ruiz de Angulo Carme Torras

Institut de Ròbotica i Informàtica Industrial (CSIC-UPC)

Edifici NEXUS, Gran Capità 2-4

08034-Barcelona. SPAIN

e-mail : vruiz@iri.upc.es ctorras@iri.upc.es

I. Introduction

The degradation in performance of an associative network over a training set when new patterns are trained isolatedly is usually called forgetting or catastrophic interference. Applications entailing the learning of a time-varying function require the ability to quickly modify some input-output patterns while at the same time avoiding catastrophic forgetting. Learning algorithms based on the repeated presentation of the learning set –back-propagation being the most popular representative– are only suited to tasks admitting two separate phases, an off-line one, for learning, and another one for operation.

If a very different and representative input-output pattern needs to be learned after the training of the main set of patterns has been completed, one gets into trouble. One can train the new pattern isolatedly or retrain a mixture of the new and old patterns. The first option is the best for this kind of applications because a quicker adaptation is obtained, but the interference must not be very strong, or at least one must be able to mitigate it. Unfortunately, the forgetting problem turns out to be a very serious drawback of back-propagation networks.

Although some interest has recently emerged on this issue, the connectionist community has not yet paid enough attention to it. Ratcliff [1] and Mc Closkey & 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 back-propagation so as to produce semi-distributed representations. In these representations, only a few hidden units take the value 1, while the majority take the value 0. This type of approach has very serious convergence problems and requires a much larger number of hidden units than straight back-propagation [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.

Hetherington and Seidenberg [5] and especially Robins [6] have studied another aspect of the problem, namely the influence of the training regimes. The later proposes to mix the training of the new pattern with pseudo-patterns, i.e, points of the function implemented by the network. The problem is that the number of pseudo-patterns needed to mix with the new one is much higher than the number of old true patterns required to get the same effect.

Brousse and Smolensky [7] 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 [8] 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. Sharkey and Sharkey (1994) used also autocoder networks in their studies of catastrophic interference. We think that results about forgetting obtained with autocoder networks and low-error patterns must not be extrapolated to more general situations.

Krusche [9] 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 [10]

[11] [12]. 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 radii). But it is precisely this locality what 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 tradeoff 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", has been 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. [13]. 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. Our approach, based on the transformation of the problem into a non-constrained optimization has evident advantages. For a more detailed comparison see [14].

II. A priori limitations

It is worth noting 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, some existing a priori limitations are spelled out below.

When a feed-forward 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 [15] [16]. 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.

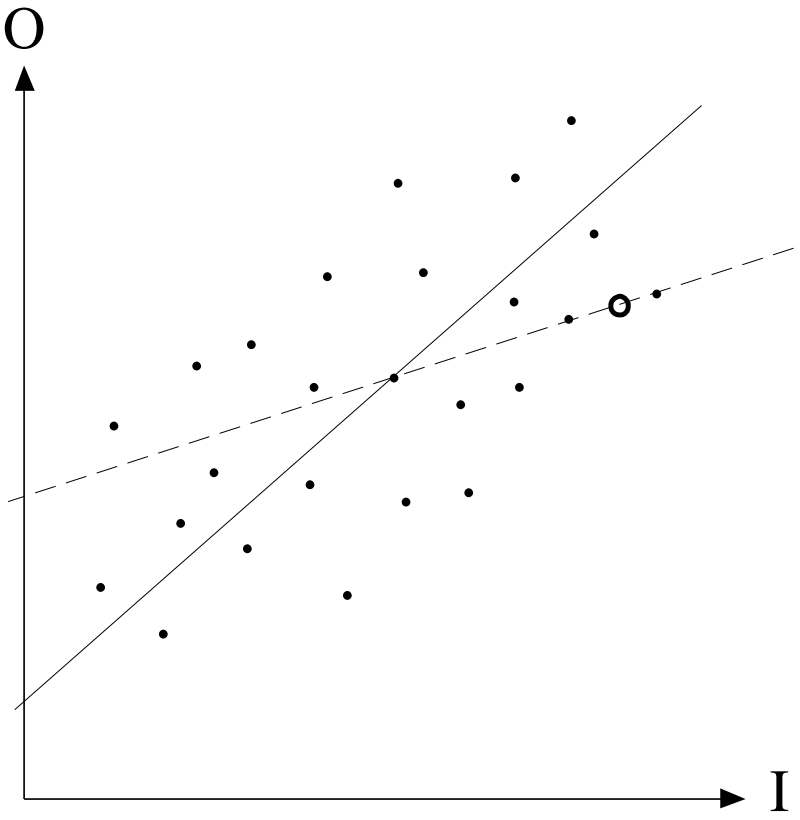

Figure 1. *The points represent the learned patterns, which are approximated by the network with the continuous line. Constraining the network to perfectly encode the new association (little circle), while minimizing the error over the old patterns, gives a global approximation (dashed line) worst than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

Now suppose that the chosen architecture is unable to encode all the patterns perfectly. Let E be the error function over the patterns 1.. n-1, E' be the error function over the patterns 1..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 Figure 1, it is possible that E' grows when one forces the surface implemented by the net 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 input-output 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 can be interpolated only 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.

The moral of this discussion is that not only it is impossible to devise a "perfect" algorithm for the non-disruptive 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

The problem addressed in this chapter can be formulated 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_i as a function of weight increments:

$$\text{Min} \ \sum_{i=1..n-1} E_i(\Delta W) \qquad (1)$$
$$E_n(\Delta W) = 0$$

Evaluating the E_i'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 truncated Taylor series expansion, for example, with a second-order polynomial in the weights. A linear model is too rude and not feasible because, as it will be pointed out at the end of Section V, it may turn the problem into an unsolvable one. The coefficients of the polynomial have a direct interpretation in terms of first and second derivatives of E. Cross-terms are not included because, besides adding great complexity to the algorithm, the calculation of the Hessian requires computations very costly in memory and time.

Thus, the most faithful problem formulation we can reasonably aspire to deal with is:

$$\text{Min} \ F(\Delta W) = \sum_i c_i \Delta w_i^2 + \sum_i b_i \Delta w_i$$
$$E_n(\Delta W) = 0 \qquad (2)$$

where F is the cost function that estimates the error increment in E (loss function) when the b_i and c_i constants are the first and second weight derivatives of E, respectively.

A usual way to tackle a constrained optimization problem is to linearly combine the cost function with the deviation of the constraint from zero, and then minimize this new error function:

$$\text{Min } \mu F(\Delta W) + \text{ß } E_n (\Delta W) .$$

In the minimization of this function, there is a tradeoff between E_n and F that depends on μ and ß, and the error in the new pattern will not be 0 unless the ß/μ relation tends to infinite with time. In practice this is impossible and it is approximated through an appropriate schedule for changing μ and ß.

The algorithm we have developed avoids this approximation by converting the constrained minimization problem into an unconstrained one, thus tackling the problem in a more direct and efficient manner. A different derivation of the algorithm can be found in [14].

IV. Transformation into an unconstrained minimization problem

The crucial finding underlying the transformation described in this section, and finally leading to the LMD algorithm, is the existence of a one-to-one mapping (except in very special cases) between a hidden-unit configuration and the best solution in a big subset of the weight increment space, $\mathbf{\Delta W}$. It is therefore possible to disregard all solutions but the best one from each subset, thus drastically reducing the search space. Furthermore, due to the one-to-one relation mentioned above, the optimum solution can be looked for indirectly by searching through the set of hidden-unit configurations.

To explain this in more detail, we need to introduce some notation. An individual weight will be called w_{ji} (from unit i to unit j). Inc(j) is the index set of the units from which unit j receives direct input.

Out(j) is the index set of the units to which unit j sends direct output. The connection graph is supposed to be loop free. Let $x_j = \sum_{i \in Inc(j)} w_{ji}\, y_i$ be the total input received by unit j, and $y_j = f_j(x_j)$ be the activation of the same unit, where f_j is the activation function of unit j, that for the moment we assume is invertible. **I** and **O** stand for the spaces of input and output vectors, while I_d and O_d denote the input vector and the output vector of the new pattern.

Remember that our actual variables are weight increments and thus we consider the current weights as constants. All possible configurations of hidden-unit activations y_j define a vector space **H**, in which each component is limited by the range of its corresponding activation function. Then we define f_1: $\mathbf{\Delta W} \longrightarrow \mathbf{H}$, the function that produces the vector of hidden-unit activations originated by a certain ΔW, when I_d is presented as input to the network. We define also f_2: $(\mathbf{\Delta W}, \mathbf{H}) \longrightarrow \mathbf{O}$ as the function that returns the network output vector originated by a set of increments and hidden-unit activations, under I_d as input to the network. Notice that this function is only defined for $(\Delta W, H)$ such that $f_1(\Delta W) = H$.

Finally, we define the one-to-one mapping we have referred to above:

$$D: \mathbf{H} \longrightarrow \mathbf{\Delta W} \qquad (3)$$

$$H \longrightarrow \Delta W = D(H) \quad \text{such that} \quad F(D(H)) = \min F(\Delta W)$$

$$f_1(\Delta W) = H$$

$$f_2(\Delta W, H) = O_d$$

D(H) is thus the best solution among those that produce the hidden activation vector H when the new pattern (I_d, O_d) is perfectly encoded. Figure 2 illustrates this mapping graphically, as well as the displacement of search from the original weight-increment space **ΔW** to the space of hidden unit configurations **H**.

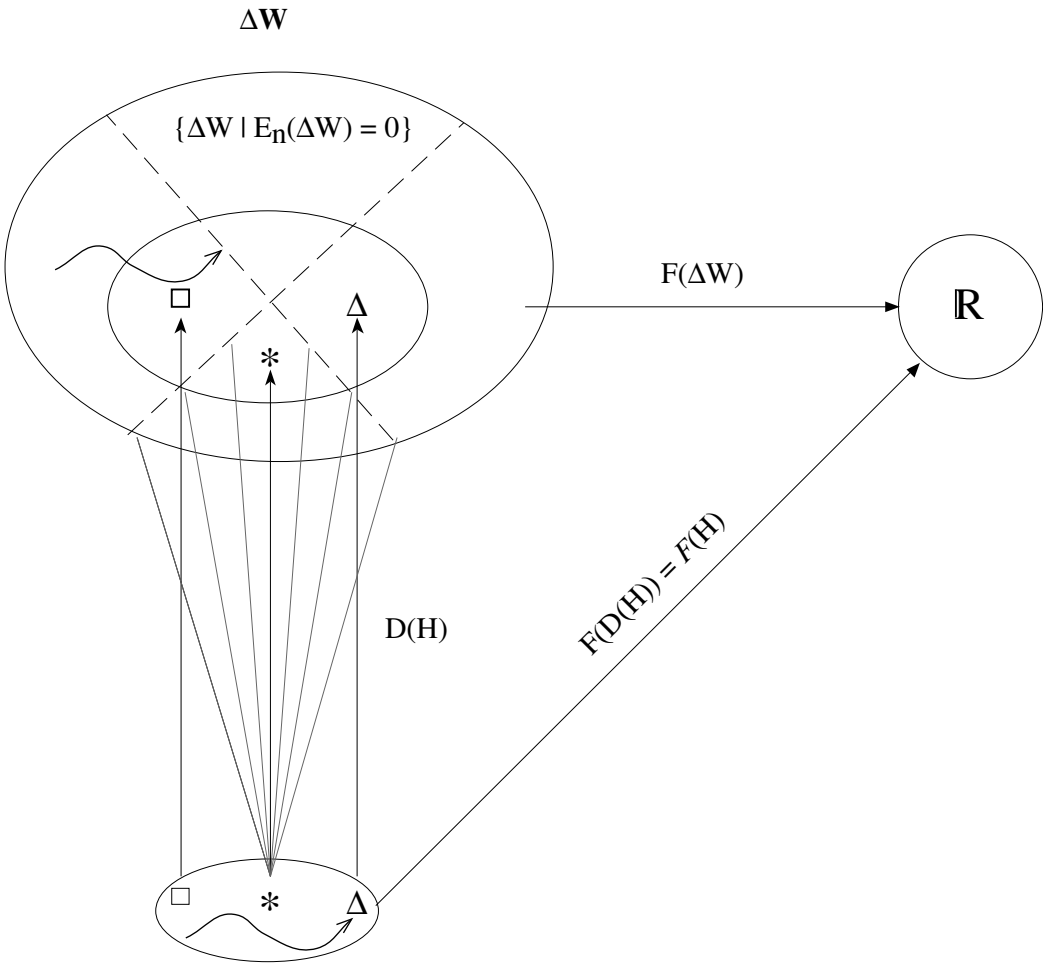

Figure 2. *Diagram illustrating the meaning of the mapping D. The dashed lines delimit the zones of* ***ΔW*** *leading to the same hidden-unit configuration in response to the new input pattern. In the intersection of one of these zones with the subset of* ***ΔW*** *satisfying the new pattern, there is a unique minimum for the function F, which is the one we place in D(****H****). Thus, only the weight increments in D(****H****) are worthwhile the search, which can be carried out through the function F(****H****) =F(D(****H****)). The search is thus brought to the space of hidden-unit configurations and, when finished, the solution obtained H* can be easily translated to its corresponding weight increment D(H*).*

Let *F* (H) denote the H-dependent function F(D(H)). Provided we derive an expression for D and prove that D is a well-defined one-to-one mapping, which we do in the next section, we have transformed the original problem into the following one:

Min *F* (H)

since, once the solution to this problem H* is obtained, the solution to the original problem is just $\Delta W^* = D(H^*)$.

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.

V. The one-to-one mapping D

The validity of the transformation performed in the preceding section depends on the fact that D is really a function, i.e., that for each H there is one and only one ΔW satisfying the conditions in (3). Let us now prove this fact.

We start by noting that the unit activations y_j are components of either H or the new input-output pattern. Due to the assumption that activation functions f_j are invertible, the x_j become fixed as $x_j = f_j^{-1}(y_j)$. Therefore, in what follows we assume that x_j and y_j are constants and the usual equations $x_j = \sum_i w_{ji} y_i$ do not hold.

Then consider:

$$F_j = \sum_i c_{ji} \Delta w_{ji}^2 + \sum_i b_{ji} \Delta w_{ji}$$

(4)

$$R_j \equiv \sum_i \left(\Delta w_{ji} + w_{ji} \right) y_i - x_j = 0$$

When j runs along all the indices such that $Inc(j) \neq \emptyset$ (hidden and output units), $F = \sum_j F_j$ and the set of R_j is equivalent to the conditions in the definition of D in (3). Thus (3) can be rewritten as:

$$\underset{\{ R_j \}_{j / Inc(j) \neq \emptyset}}{\text{Min}} \sum_{j / Inc(j) \neq \emptyset} F_j \ (\Delta W) \qquad (5)$$

Since F_j *and* R_j *share all variables and have no one in common with* F_k *and* R_k *if* $k \neq j$, *our problem is reduced to independently minimize each* F_j *under the* R_j *constraint.*

This can be easily done, for example, using the Lagrange multipliers method. First we put R_j in a more convenient manner:

$$R_j \equiv \sum_i \Delta w_{ji} \ y_i = x_j - \sum_i w_{ji} \ y_i$$

and calling Δx_j the constant $x_j - \sum_i w_{ji} y_i$, we have:

$$R_j \equiv \sum_i \Delta w_{ji} \ y_i = \Delta x_j \ .$$

(6)

Now we define the functions to be minimized G_j as:

$$G_j = F_j + t \, R_j \ .$$

All $\partial G_j / \partial \Delta w_{jk}$ must be zero in the solution,

$$\frac{\partial G_j}{\partial \Delta w_{jk}} = 2\, c_{jk}\, \Delta w_{jk} + b_{jk} + t\, y_k = 0$$

hence,

$$\Delta w_{jk} = \frac{-b_{jk} - t\, y_k}{2\, c_{jk}} \quad .$$

(7)

Then using the constraint R_j in (6)

$$\Delta x_j = \sum_i \Delta w_{ji}\ y_i \ = -\sum_i \frac{b_{ji}}{2\, c_{ji}}\, y_i \ - \frac{t}{2} \sum_i \frac{y_i^2}{c_{ji}} \quad ,$$

therefore,

$$t = -2\, \frac{\Delta x_j + \sum_i \frac{b_{ji}\ y_i}{2\, c_{ji}}}{\sum_i \frac{y_i^2}{c_{ji}}} \quad ,$$

and substituting in (7),

$$\Delta w_{jk} \ = \ \frac{y_k \ \left(\Delta x_j + \sum_i \frac{b_{ji}\ y_i}{2\, c_{ji}} \right)}{c_{jk} \sum_i \frac{y_i^2}{c_{ji}}} \ - \ \frac{b_{jk}}{2\, c_{jk}}$$

(8)

Observe that for the case in which $c_{ji} = 1$, $b_{ji} = 0$, this is the Widrow-Hoff rule [17]. This formula will be simplified further, but we can conclude now that D 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_{ji} are non zero. We have to take care of this when selecting the parameters of the quadratic function (see Section VIII). This is the reason for the unsuitability of a linear F.

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

VI. The LMD algorithm

We can now derive a gradient algorithm for carrying out the minimization in (5). Let us begin by giving $F = \Sigma_j F_j$ a more explicit form. From (8):

$$\Delta w^*_{jk}(Y) = \frac{E_j}{c_{jk} M_j} y_k - \frac{b_{jk}}{2 c_{jk}} ,$$

where $E_j = f_j^{-1}(y_j) - \sum_i w_{ji} y_i + \sum_i \frac{b_{ji} y_i}{2 c_{ji}}$ and $M_j = \sum_i \frac{y_i^2}{c_{ji}}$.

Substituting this expression for Δw^*_{jk} into (5):

$$F_j(Y) = \sum_i c_{ji} \left(\frac{E_j}{c_{ji} M_j} y_i - \frac{b_{ji}}{2 c_{ji}} \right)^2 + \sum_i b_{ji} \left(\frac{E_j}{c_{ji} M_j} y_i - \frac{b_{ji}}{2 c_{ji}} \right) =$$

$$= \sum_i c_{ji} \frac{E_j^2 y_i^2}{c_{ji}^2 M_j^2} + \sum_i c_{ji} \frac{b_{ji}^2}{4 c_{ji}^2} - 2 \sum_i c_{ji} \frac{E_j y_i}{c_{ji} M_j} \frac{b_{ji}}{2 c_{ji}} + \sum_i b_{ji} \frac{E_j y_i}{c_{ji} M_j} - \sum_i \frac{b_{ji}^2}{2 c_{ji}} = \sum_i \frac{E_j^2 y_i^2}{c_{ji} M_j^2} - \frac{1}{4} \sum_i \frac{b_{ji}^2}{c_{ji}} ,$$

where the definition of M_j was used in the last step. Observe that M_j and E_j can be taken out from the sumatories:

$$F_j(Y) = \frac{E_j^2}{M_j^2} \sum_i \frac{y_i^2}{c_{ji}} - \frac{1}{4} \sum_i \frac{b_{ji}^2}{c_{ji}} = \frac{E_j^2}{M_j} - \frac{1}{4} \sum_i \frac{b_{ji}^2}{c_{ji}}$$

so finally,

$$F(Y) = \sum_j F_j(Y) = \sum_j \frac{E_j^2}{M_j} - \frac{1}{4} \sum_{j,i} \frac{b_{ji}^2}{c_{ji}} \quad . \tag{9}$$

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

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:

$$\Delta w'^{*}_{jk} = (w_{jk} + \Delta w^{*}_{jk}) - w'_{jk} = \Delta w^{*}_{jk} + \frac{b_{jk}}{2\, c_{jk}} = \frac{E_j\, y_k}{c_{jk}\, M_j} \quad . \tag{10}$$

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 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:

$$\frac{\partial F}{\partial x_j} = -2 \frac{E_j}{M_j} + 2 \; f_j'(x_j) \sum_{s \in \mathrm{Out}(j)} \frac{E_s\, w'_{sj}\, M_s + E_s^2\, y_j / c_{sj}}{M_s^2} \quad . \tag{11}$$

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 back-propagation. In the first forward phase, E_j and 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 back-propagation 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 M_j and 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.

VII. Adaptation of LMD for RBF units.

Section II demonstrated that the forgetting problem can be unsurmuntable for fixed architectures in some applications. In this case it would be convenient to use incremental architectures, and RBF units seem the best candidates to be the new units, because they are suitable to modify locally the learned function. In the present section the derivations needed to apply the LMD algorithm to RBF units are spelled out.

We shall limit the study to the case of $b_{ji} = 0$ which, as we will see in the next section, is the one used in practise, since the inclusion of the b_{ji} reduces to a preprocessing step with a standard second-order algorithm. The solution for arbitrary c_{ji} is difficult to find analytically. For this reason, the c_{ji} of the weights impinging on a gaussian unit j are all taken to be equal and are denoted by c_j.

The propagation function of the weights outcoming from a RBF unit is a scalar product and the activation function of the output units is linear or sigmoidal. Thus, the cost function associated to these weights is the same as in back-propagation networks.

RBF units, however, introduce an element that does not fit in the solution framework stated in Sections IV and V. The activation function of these units is gaussian, and thus not invertible, which was one of the assumptions made to show that D is a one-to-one function. Nonetheless, this difficulty can be easily overcome. Although we base all the framework in the hidden-unit activations for the sake of clarity in the exposition, it is also possible to base it on the total inputs to the hidden units, being D a function of these, in such a way that the invertibility assumption can be dropped.

Consider C_j and R_j when j is a RBF unit:

$$C_j = \sum_i c_j \Delta w_{ji}^2$$

$$R_j \equiv \sum_i \left((\Delta w_{ji} + w_{ji}) - y_i \right)^2 - a_j^2 = 0$$

The independence of the problems associated to each pair C_j and R_j still holds, due to the absence of common variables. Thus, it suffices to solve each subproblem using the Lagrange multipliers. The function G_j that must be optimized is again $G_j = C_j + t\, R_j$, and thus its derivatives must be null:

$$\frac{\partial G_j}{\partial \Delta w_{jk}} = 2\, c_j\, \Delta w_{jk} + 2\, t\, (w_{jk} + \Delta w_{jk} - y_k) = 0 \ ,$$

hence

$$\Delta w_{jk}^* = \frac{-\, t\, (w_{jk} - y_k)}{c_j + t} \ .$$

Now we substitute the increments in R_j to obtain t:

$$\sum_i \left((w_{ji} - y_i) + \frac{-\, t\, (w_{ji} - y_i)}{c_j + t} \right)^2 - a_j^2 = 0,$$

$$\left(1 - \frac{t}{c_j + t} \right)^2 \sum_i (w_{ji} - y_i)^2 = a_j^2,$$

$$\left(1 - \frac{t}{c_j + t} \right)^2 = \frac{a_j^2}{\sum_i (w_{ji} - y_i)^2} \ .$$

From this we cannot obtain t easily. We shall do the variable change $t' = -t / (c_j + t)$. In this way, $\Delta w_{jk}^* = t' (w_{jk} - y_k)$ and

$$1 - \frac{t}{c_j + t} = \pm \frac{a_j}{\sqrt{\sum_i (w_{ji} - y_i)^2}} \qquad \Rightarrow \qquad t' = \pm \frac{a_j}{\sqrt{\sum_i (w_{ji} - y_i)^2}} - 1 \ ,$$

so

$$\Delta w_{jk}^* = (w_{jk} - y_k) \left(\pm \frac{a_j}{\sqrt{\sum_i (w_{ji} - y_i)^2}} - 1 \right) = \pm \frac{(w_{jk} - y_k)\, a_j}{\sqrt{\sum_i (w_{ji} - y_i)^2}} - (w_{jk} - y_k).$$

The increments Δw^*_{jk} can take two values. We must select those corresponding to the minimum of C_j. Since a_j is always positive, it is easy to check that

$$\Delta w^*_{jk} = \frac{(w_{jk} - y_k)\, a_j}{\sqrt{\sum_i (w_{ji} - y_i)^2}} - (w_{jk} - y_k)$$

is always the option with lower absolute value, and thus, this is the desired solution.

Let us now calculate C_j:

$$C_j = c_j \sum_k \left(\frac{(w_{jk} - y_k)\, a_j}{\sqrt{\sum_i (w_{ji} - y_i)^2}} - (w_{jk} - y_k) \right)^2 =$$

$$= \frac{c_j\, a_j^2 \sum_k (w_{jk} - y_k)^2}{\sum_i (w_{ji} - y_i)^2} + c_j \sum_k (w_{jk} - y_k)^2 - \frac{2\, c_j\, a_j \sum_k (w_{jk} - y_k)^2}{\sqrt{\sum_i (w_{ji} - y_i)^2}} =$$

$$= c_j\, a_j^2 + c_j \sum_k (w_{jk} - y_k)^2 - 2\, c_j\, a_j \sqrt{\sum_k (w_{jk} - y_k)^2}\,.$$

The gradient of C_j with respect to a_j is:

$$\frac{\nabla C_j}{\nabla a_j} = -2\ c_j\, a_j + 2\, c_j \sqrt{\sum_k (w_{jk} - y_k)^2}$$

We can finally write the total gradient of a RBF unit with respect to C. Remember that the second term of (11) corresponds to the gradients of the outcoming weights, which are the same in this case:

$$\frac{\nabla C}{\nabla a_j} = -2\ c_j\, a_j + 2\, c_j \sqrt{\sum_k (w_{jk} - y_k)^2} + 2\ f_j'(a_j) \sum_{s \in \mathrm{sal}(j)} \frac{E_s\, w'_{sj}\, M_s + E_s^2\, y_j / c_{sj}}{M_s^2}$$

Since each RBF unit is directly connected to the input of the network, $y_k = x_k$, and thus $\sqrt{\sum_k (w_{jk} - y_k)^2}$ (which would be a_j in the forward operating mode of the network) is constant during the search and must be computed only once .

VIII. 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, $c_{jk} = \partial^2 E/\partial w^2_{jk}$ and $b_{jk} = \partial E/\partial w_{jk}$.

With this choice, F is exactly the local estimation of E assumed by several authors [18], [19], [20] to justify this simple pseudo-Newton rule for optimizing E:

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

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 application of LMD. Later in Section XI, 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:

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

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 $\sum_{jk} c_{jk} \Delta w^2_{jk}$, this choice of coefficients is in average the best for estimating E.

In order 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 behaviour 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.

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 1 and all the b_{jk} equal to 0, 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,.. ,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 an Sejnowsky [22].

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

IX. 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 1. 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 $\Delta F \geq 0$

$x_j \leftarrow x_j - (1- \mu^-)\ \mu \dfrac{\partial F}{\partial x_j}$

$\mu \leftarrow \mu\ \mu^-$

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 E_j and 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 2 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 :

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

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.

X. 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..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 *back-propagation time*, because these are the learning equations of back-propagation as a continuous dynamical system. To estimate back-propagation time we constrain pure gradient back-propagation 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:

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

(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:

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

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:

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

Therefore, the time to complete a trajectory is $\Sigma_k\ \mu^{k)}$ 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 back-propagation 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 back-propagation 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:

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

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(ΔW, $\partial E/\partial W$ (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. μ 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 μ near the highest values allowed by the angle continuity constraint. In our simulations, $\alpha = 0.5$ and $\beta = 1.2$. To compute ang(ΔW, $\partial E/\partial W(W)$), 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.

XI. Experimental results

A. Scaling properties

Figure 3 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 gotten by multiplying by 10, 20 ... the number of units in each layer of this network. The initial μ for the 8-3-1 network was 1, 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 $F=\sum \Delta w^2_{jk}$ and each weight w_{jk} in the network was obtained randomly with the uniform distribution [-1.5 / |Inc (j)|, 1.5 / |Inc (j)|].

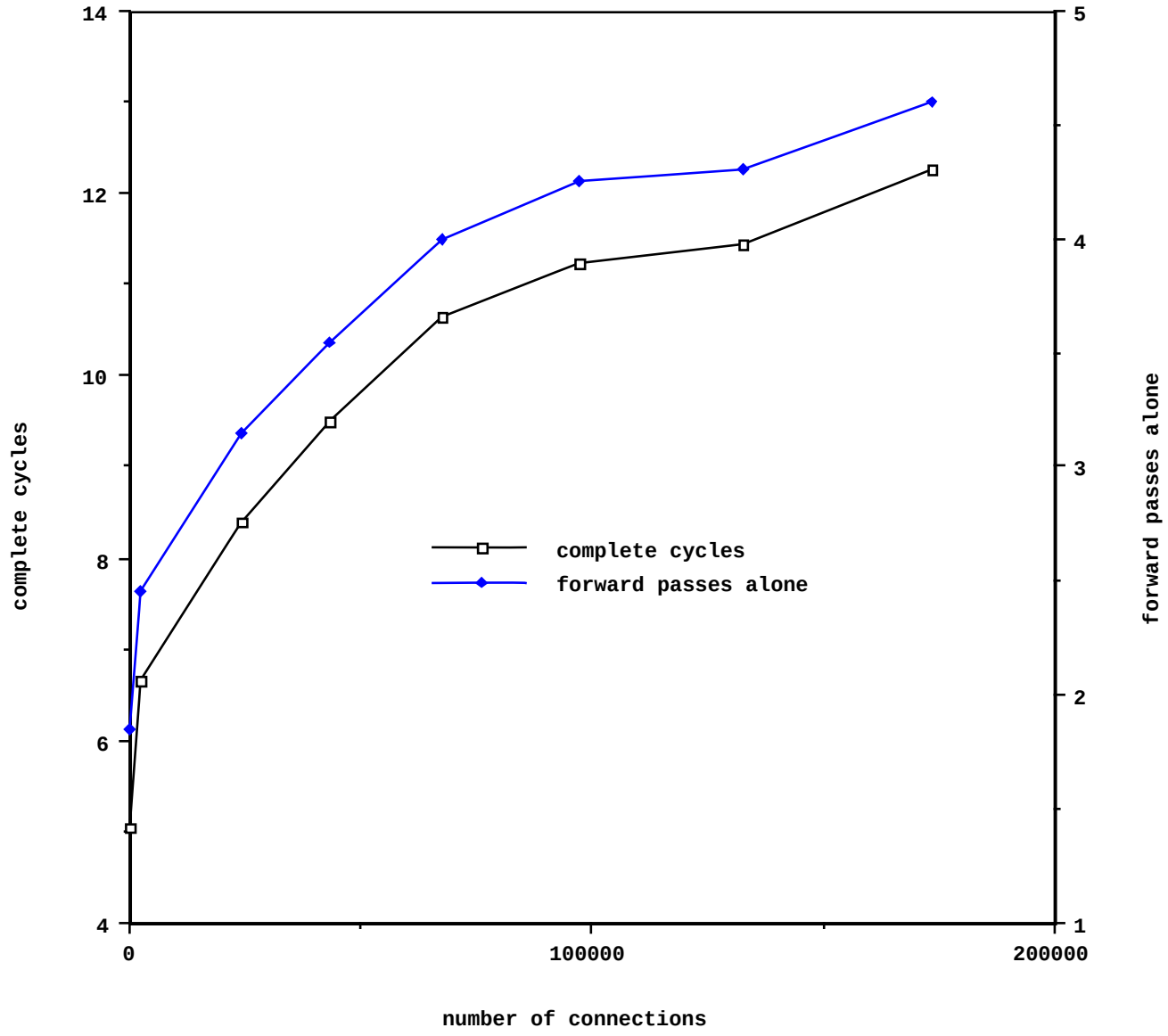

Figure 3. *Scaling experiment: Number of LMD complete cycles and feedforward steps alone (due to overshot steps) as a function of network size (number of connections).*

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 back-propagation.

B. Solution quality for different coefficients settings

Table 1 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 $c_{jk} = |\partial^2 E / \partial w^2_{jk}|$. The error measure used is

$$E = \frac{1}{2} \frac{1}{n\ No} \sum_{\substack{j=\ 1..n \\ i=\ 1..No}} \varepsilon^2_{ji} \ ,$$

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 8 input units, 4 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 twenty 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.

Table 1. Error increments after applying LMD with $c_{jk} = 1$ and $c_{jk} = |\partial^2 E/\partial w^2_{jk}|$.

Architecture	**8 - 3 -1**		**32 - 12 - 4**		**300 - 30 - 10**	
Nr. of Patterns	**5**	**10**	**10**	**25**	**50**	**150**
Coarse MDL	0.064	0.111	0.018	0.032	0.002	0.002
Standard MDL	0.038	0.087	0.012	0.028	0.001	0.002

Table 2. Error increments out of the minimum.

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

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 2 presents results for the 8-3-1 architecture trained with 5 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. 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.

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 $W^*_{1..n-1}$ be the weight configuration at

which the minimum error $E^*_{1..n-1}$ for the set of patterns 1..n-1 is attained, and E be the error function over patterns 1..n. For each network and new pattern, we first calculate very accurately the minimum error E* for the set of patterns 1..n, then we measure curve length from $W^*_{1..n-1}$ to the first point in the trajectory with an error less than E* + .2 ($E(W^*_{1..n-1})$ - 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 3 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 back-propagation 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. However, this is a very abnormal situation 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 Figure 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.

Table 3. Length travelled with and without LMD to reach the minimum of the patterns 1,..,n.

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

D. LMD versus back-propagation

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

In Figure 4a the net 32-12-4, trained with 10 random patterns, undergoes the introduction of a new pattern with different back-propagation 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 $\mu \to 0$ can only be approximated with large computational costs. Figure 4b displays how the distances to the starting point showed in Figure 4a affect the error in the old patterns, and how this in turn affects the recovery time (measured in terms of back-propagation time) needed to relearn both the new and the old patterns. The distance lower bound for back-propagation when $\mu \to 0$ (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.

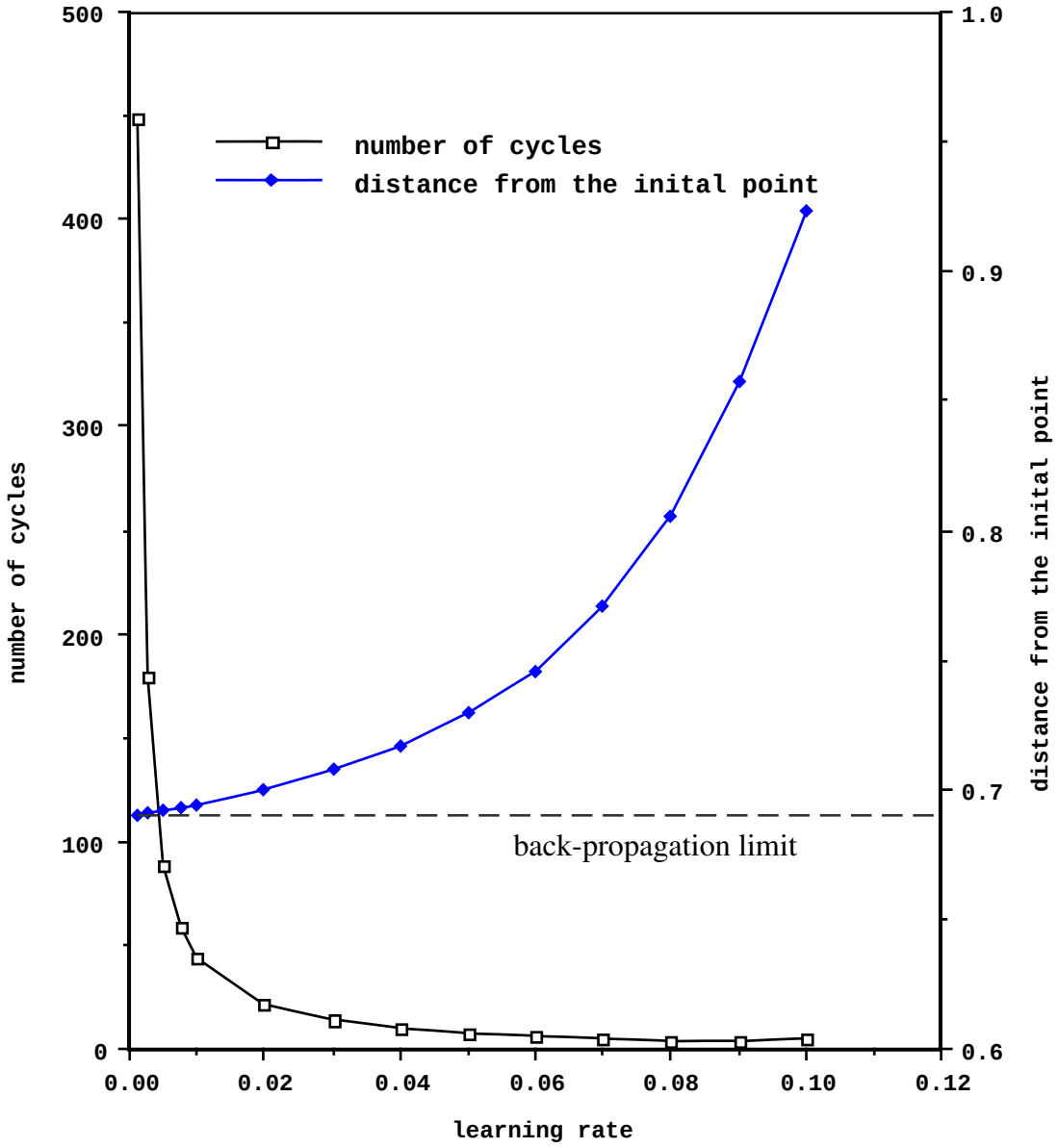

Figure 4a. *Starting with a network in the minimum of the error function for the old patterns, a new pattern is introduced with different back-propagation learning rates. The graphic shows the number of cycles needed to learn the pattern and the total distance covered from the initial point in each case. It is clear that, when the learning rate tends to zero, the distance tends to a limit.*

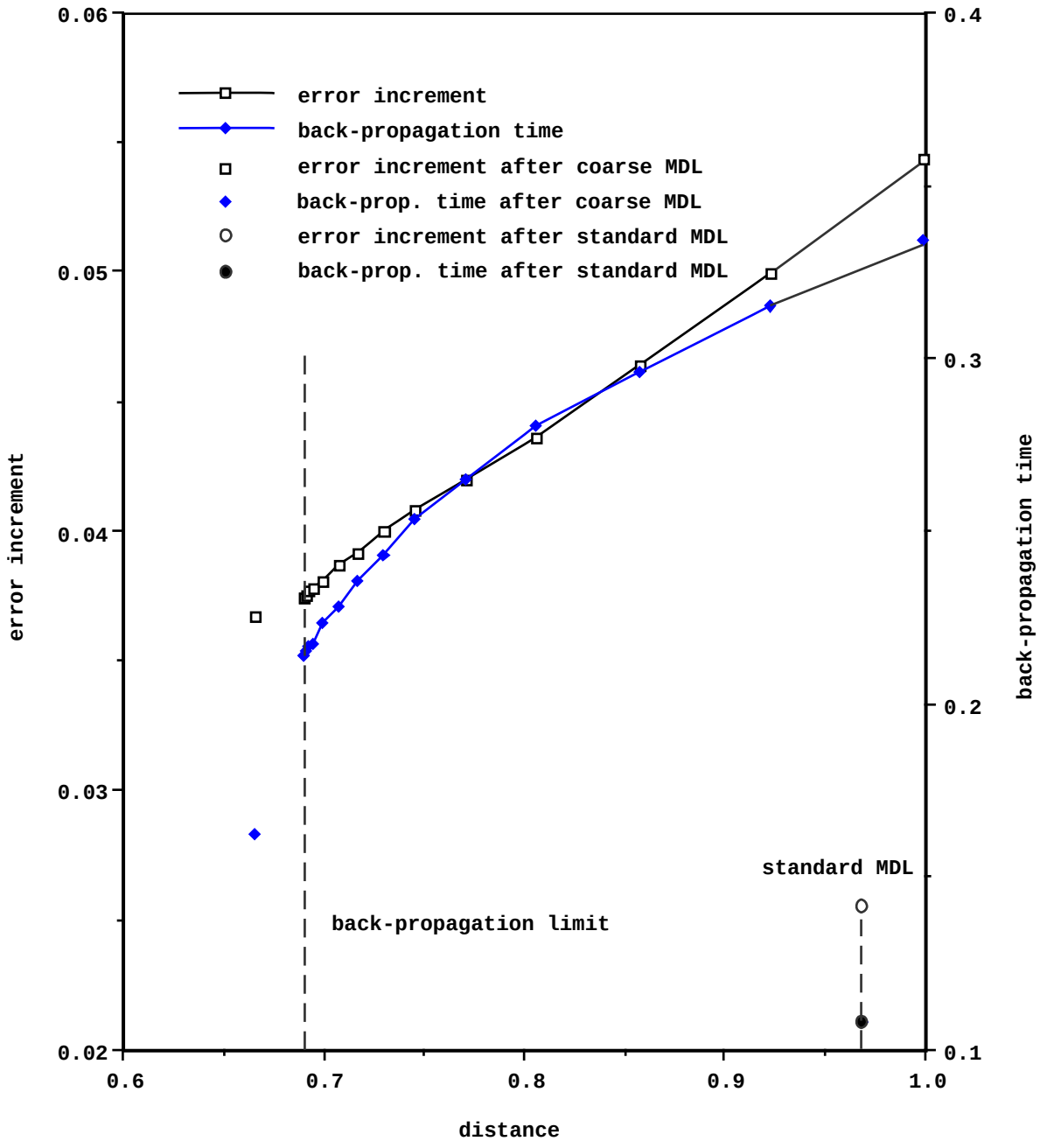

Figure 4b. *Another aspect of the experiment described in Figure 3a. The distances represented in Figure 3a are now in the axis of abcises. The two curves represent the error increments and the back-propagation times beginning after the introduction of the new pattern and ending in the global minimum 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 back-propagation limit. However, standard LMD ($c_{jk} = |\partial^2 E/\partial w^2_{jk}|$), which does not take into account distance, gets the best results.*

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 however 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.

XII. Discussion

A. The influence of the back-propagation advance rate on forgetting

One of the main results we have obtained is the dependence of back-propagation forgetting on the learning rate. Cohen [2] already observed this in a systematic variation of all back-propagation 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 back-propagation 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 back-propagation 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 tradeoff in back-propagation, thus finding another use complementary to the one for which it was designed in Section IX. 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 back-propagation limit- and only 27 cycles were used -almost half of those needed by back-propagation with the appropriate μ to get the same distance-. The algorithm to compute back-propagation 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 XI, D, as well as the reasoning about the convenience of dispensing with the b_{jk} parameters presented in Section VIII, is that one must wait as long as possible to the completion of the learning of the previous patterns (by second-order, standard back-propagation 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, in order 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 Figures 5a and 5b, it can be seen that, when the training is finished, the level of error is very good, practically 0, 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.

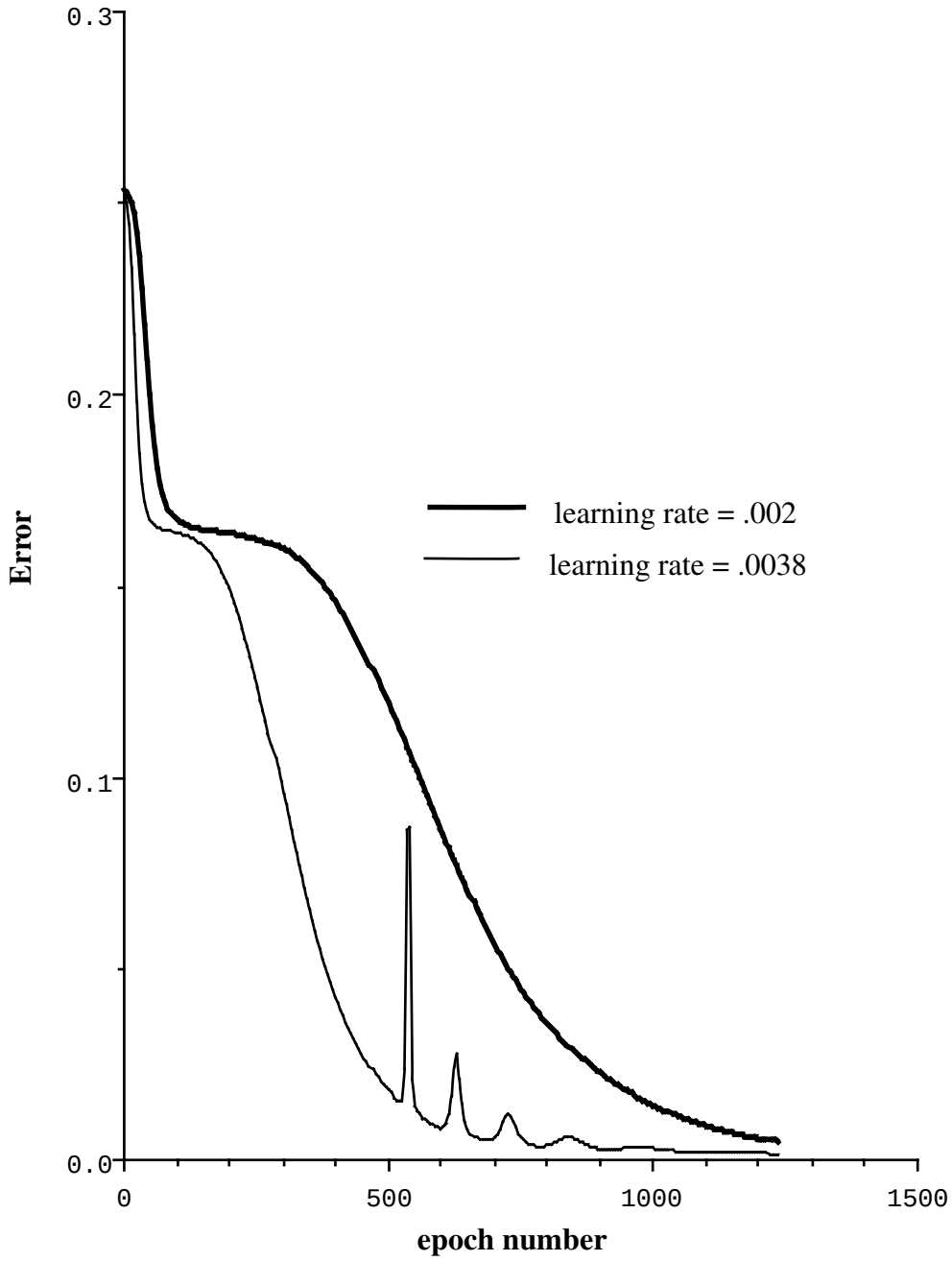

Figure 5a. *Error evolution in a network with two different learning rates.*

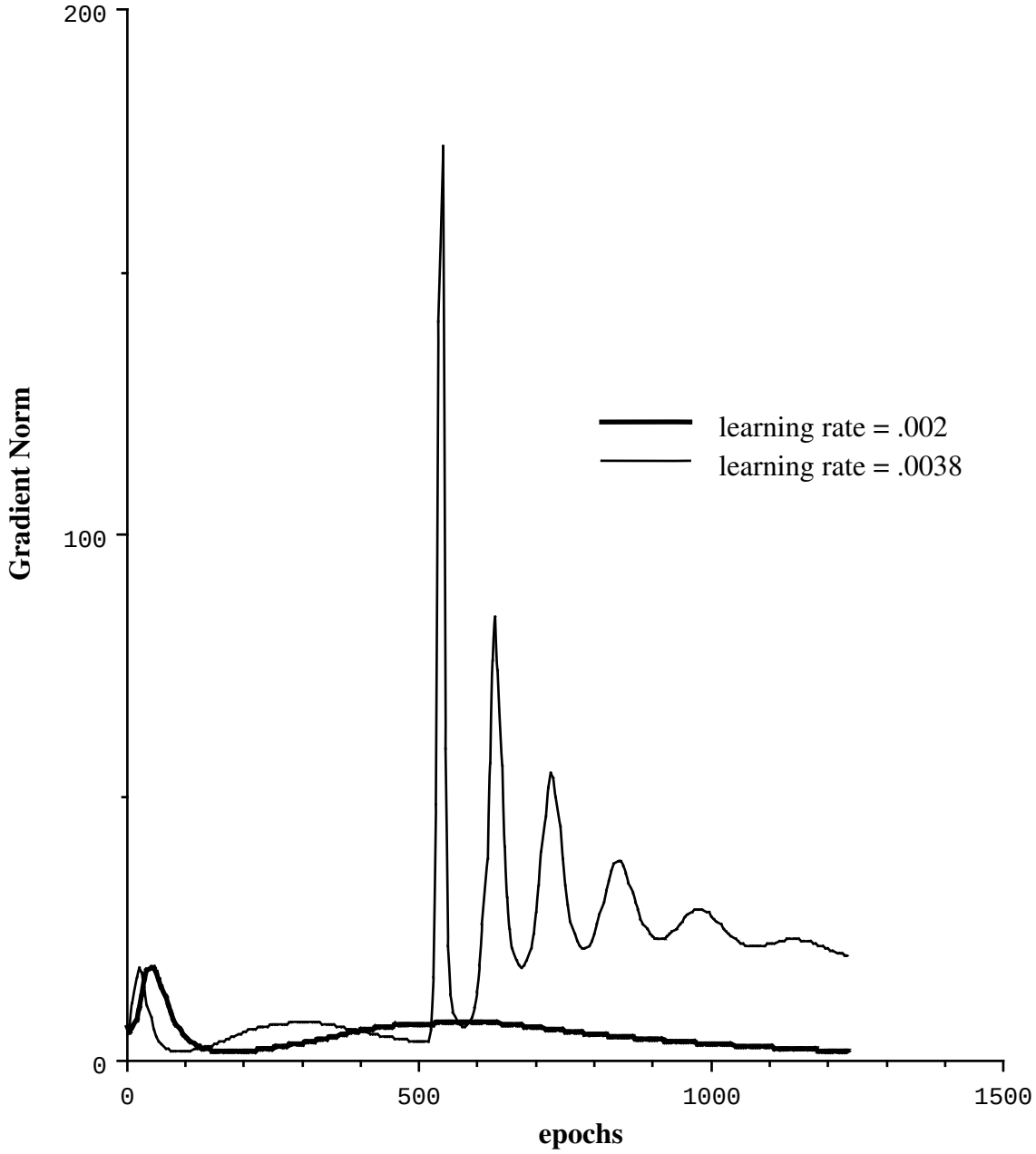

Figure 5b.*Gradient norm evolutions along the learning corresponding to the training curves in Figure 5a.*

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 back-propagation 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 back-propagation with a very cautious learning rate or, more efficient and safe, one can use one of the algorithms presented in Section IX, 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].

XIII. Conclusion

There are easy ways to overcome the catastrophic interference problem by using incremental architectures. In fact, a local hidden unit can be added each time an erroneous new pattern arrives, centering its receptive field in the input pattern and assigning an appropriate value to the hidden-output

weight in order to correct the error. It is enough to make the receptive field as little as necessary to mitigate interference up to a desired degree.

However, there are reasons to develop methods compatible with fixed architectures:

1) In an always changing environment, the network would tend to grow indefinitely.

2) More importantly, even when adding a new unit, the new information should be somehow shared with the rest of the network. Otherwise, if each unit only assumes responsibility for a point, one gets a kind of look-up table network. When adding a new unit is considered necessary, it is advisable to correct the new pattern as much as possible with the rest of the network by applying the methods developed to minimize damage over the previous patterns in the case of fixed architectures. For this reason, it is important that these methods are compatible with the use of local units.

The problem of catastrophic interference can be divided into two subproblems. The first (retroactive interference) arises when one tries to introduce new information in a previously trained network, without disrupting the stored information. The second is how to encode a learning set of patterns in such a way that the retention of this set is maximized when some new unknown information has to be stored. The second problem is tackled in [26, 21], where its relation with other connectionist problems is discussed.

This chapter has been concerned with the retroactive interference subproblem, which is solved with the LMD algorithm, based on the relation found between the constrained minimization that represents this problem, and the unconstrained minimization of an implicit function over the hidden-unit activations. Full parallelism, both within and between layers, is one of the features of LMD. We implemented two versions of LMD, the coarse one, which searches for the nearest solution in the weight space, and the standard one, based on the information provided by the second derivatives of

the weights. We have demonstrated the good scaling properties of the algorithm, as well as the solution quality and computational savings derived from its application. The experiments also revealed that when the learning rate tends to zero, back-propagation approximates the coarse version of LMD. The latter algorithm can control the comparative importance of the previous data by weighing each pattern through the coefficients c_{jk} in the error function, i.e., using $c_{jk} = \sum_p \alpha(t_p) \frac{\partial^2 E_p}{\partial w_{jk}^2}$, where $\alpha(t_p)$ is the weight of each pattern p, depending on the time elapsed since the pattern was presented to the network for the first time. 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 evaluating the goodness of some initial weights for faster learning [27].

The variety of stratregies possible for the use of LMD is worth of further study. A less raw application of LMD could be beneficial or even necessary, as Section II suggests. The question is what fraction of the new pattern error should be reduced with LMD? The answer when tackling the minimization of E(W) will depend on the number of patterns already stored and on the average of error increment expected to be required to modify the network outputs for the pattern, but anyway the new pattern should not be forced to be less erroneous than the previous patterns. On the other hand, normally we envisage to optimize generalization, which provides another constraint: the error should not be reduced beyond the variance of the noise of the output data. LMD could also be combined with back-propagation, by applying it in each back-propagation cycle to the most erroneous pattern, which would help to avoid local minima and, as we are beginning to experience, seems to favour learning enormously.

Finally, a general-purpose algorithm based on LMD can be devised. This algorithm would benefit from direct manipulation of the hidden-unit representations, just as the algorithms presented in [28], [29] and [30] do. The rationale is that, since LMD can exactly control the network output for each pattern, it is possible to implement a gradient descent algorithm at this level. If these variables were independent, due to the quadratic shape of E(W) with respect to the output of the network for the training patterns, the algorithm would bring the network to the minimum in a single step. However, the network outputs for the different patterns are dependent and, although LMD allows to change some of them while minimizing the effect on the others, shorter steps should be taken. Let Y_p be the network output for pattern p, then the fraction of $(O_p - Y_p)$ reduced in each cycle would be regulated by a parameter general for all patterns, that should be enlarged or reduced depending on how much the result of applying LMD to all patterns in the previous step deviated from the straight line (in network output space) leading to the minimum.

Acknowledgement

Carme Torras acknowledges partial support from the Commision of the European Union under contract No. 8556 (NeuroColt).

References

[1] R. Ratcliff, "Connectionist models of recognition memory: constraints imposed by learning and forgetting functions", Psychological Review 97, 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.). Academic Press, New York, 1989.

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

[4] J.M.J. Murre, chapter 12 of "Categorization and learning in neural networks", Ph.D Thesis, University of Leiden, 1991.

[5] P.A. Hetherington and M.S. Seidenberg, “Is there catastrophic interference in connectionist networks?”, Proceedings of the Eleventh Annual Conf. of the Cognitive Science Society, Hillsdale, N.J., pp 26-33. Lawrence Erlbaum, 1989.

[6] A. Robins,“Catastrophic forgetting, rehearsal and pseudorehersal”, Connection Science 7, pp. 123-146 (1995).

[7] O.P. Brouse and P. Smolensky, "Virtual memories and massive generalization in connectionist combinatorial learning", Proceedings of the Eleventh Annual Conference of the Cognitive Science Society, Hillsdale NJ, pp. 380-387 . Lawrence Erlbaum, 1989.

[8] P.A. Hetherington, "The Sequential Learning Problem in Connectionist Networks", Master Thesis, Department of Psychology McGill University, Montreal, Quebec, Canada, November 1990.

[9] J.K. Kruschke, “Human category learning: Implications for backpropagation models”. Connection Science 5, pp. 3-36 (1993).

[10] J. Moody and C. Darken, "Fast learning in networks of localy tuned processing units", Neural Computation 1, pp. 281-294 (1989).

[11] T. Poggio and F. Girosi, "Networks for approximation and learning", Proceedings of the IEEE 78, No. 9, pp. 1481-1497 (1990).

[12] J. Platt, "A resource-allocating network for function interpolation", Neural Computation 3, pp. 213-225, (1989).

[13] D.C. Park, M. A. El-Sharkawi and R.J. Marks II, "An adaptively trained neural network", IEEE Transactions on Neural Networks 2, pp. 334-345 (1991).

[14] V. Ruiz de Angulo and C. Torras, “On-line learning with minimal degradation in feedforward networks”, IEEE Transactions on Neural Networks 5, pp. 657-668 (1995).

[15] A. Blum and R.L. Rivest, "Training a 3-node neural network is NP-complete", Proceedings of the 1988 Workshop on Computational Learning Theory, pp. 9-18, San Mateo, CA. Morgan-Kauffman Publishers, 1988.

[16] S. Judd, "Neural Network Design and the Complexity of Learning". MIT Press, Cambridge, Massachusetts, 1990.

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

[18] S. Becker and Y. Le Cun, "Improving the convergence of back-propagation learning with second order methods", Proceedings. of the 1988 Connectionist Models Summer School (D. Touretzky, F. Hinton, and T. Sejnowski, eds.), pp. 29-37. San Mateo, Morgan Kauffman, 1988.

[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.), pp. 309-323. Norwood: Ablex, 1988.

[20] L.P. Ricotti, S. Ragazzini and G. Martinelli, "Learning word stress in a suboptimal second order back-propagation neural network", Proceedings of the IEEE Int. Conf. on Neural Networks, San Diego, Vol. I, pp. 355-364. IEEE, New York (1990).

[21] V. Ruiz de Angulo, "Interferferencia catastrófica en redes neuronales: Soluciones y relación con otros problemas del conexionismo", PhD. Dissertation, Bask Country University (1996).

[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.). MIT press, Cambridge, MA, 1986.

[23] J. Mijhuis, K. Hofflinger, V.S. Haik and L. Spaanenburg, "Limits to the fault-tolerance of a feedforward neural network with learning", Proceedings of the 20th International Symposium on Fault Tolerance Computing, Newcastle upon Tyne, pp. 228-235 (June 1990).

[24] Y. Le Cun, J. S. Denker and S. A. Solla, "Optimal Brain Damage", in "Advances in Neural Information Processing Systems" (David S. Touretzky ed.). Morgan Kauffman Publishers, 1990.

[25] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks", IEEE Transactions on Neural Networks 1, pp. 239-242 (June 1990).

[26] V. Ruiz de Angulo and C. Torras., “Random Weights and Regularization”, Proc. of the Int. Conf. on Artificial Neural Networks (Marinaro G. y Morasso P. eds.), pp. 1456-1459, Springer-Verlag, 1994.

[27] S. Gavin, "Designing multilayer perceptrons from nearest-neighbour systems", IEEE Transactions on Neural Networks 3 (March 1992).

[28] T. Grossman, R . Meir and E. Domany, "Learning by internal representations", Complex Systems 2, pp. 555-575 (1988).

[29] A. Krogh, C.J. Thorbergsson and J.A. Hertz, "A cost function for internal representation", in “Advances in Neural Information Processing Systems” (S. Touretzky ed.). Morgan Kauffman Publishers, 1990.

[30] R. Rohwer,"The moving target training algorithm", in "Advances in Neural Information Processing Systems" (D. S. Touretzky ed.). Morgan Kauffman Publishers, 1990.