MINIMALLY DISTURBING LEARNING

V. Ruiz de Angulo

Commission of the European Communities
Joint Research Centre - Ispra Site
Institute for Systems Engineering and Informatics
Neural Networks Laboratory
21020 Ispra (Va), Italy
e-mail: ruizdeangulo@ispgro.cern.ch

Carme Torras

Institut de Cibernètica (CSIC-UPC)
Diagonal, 647
08028 Barcelona, Spain
e-mail: torras@ic.upc.es

Abstract

We have undertaken the study of methods for avoiding catastrophic forgetting in feedforward neural networks, without sacrificing the benefits of distributed representations. We formalize 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 we transform this constrained optimization problem into an unconstrained one. This new formulation naturally leads to an algorithm for solving the problem, which we call Minimally Disturbing Learning (MDL). Some experimental comparisons of the performance of MDL with back-propagation are provided which, besides showing the advantages of using MDL, reveal the dependence of forgetting on the learning rate in back-propagation.

1. Introduction

In the last years great advances are taking place in the field of learning with multilayer networks. New algorithms have been developed which have proved to be able to tackle some interesting problems [Torras 1989]. The most popular and representative of them are back-propagation [Rumelhart et al. 1986] and its variants. However, for a widespread use of neural networks in real world applications, some lacks and drawbacks must be overcome. This paper deals with one of these often noticed, but almost never attended. Algorithms of the back-propagation type are not well suited for applications where learning cannot be confined to a previous phase. Generally the procedure with these algorithms is to train the network with a representative set of input-output patterns up to a certain criterium, and then make real use of the network in a separate stage. 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.

There are two possibilities:

- To train the network with the new pattern isolatedly. This will 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 period of time.

- To retrain the network directly with the new and the old patterns together. The net will not suffer from catastrophic forgetting, but a correct response to the new pattern will be available only after a long time. Even more, the time to recover similar degrees of performance over the learning set increases greatly with the growth of this set.

It seems clear that to succeed in an application of this kind it is necessary to mitigate forgetting. Up to our knowledge, very few works have tackled this issue. Ratcliff (1990), after a lot of systematic studies, simply

arrives to the conclusion that this problem cannot be satisfactorily solved. French (1991) tries to reduce its effects by modifying back-propagation to produce semi-distributed representations. He claims that the cause of forgetting is the overlap between the representations of the different patterns. But reducing the distributedness of the representations has also the very undesirable effect of losing some of the more interesting neural networks properties, like generalization (as French himself points out) and damage resistency.

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 previously trained items*. An algorithm, which we call MDL for "minimally disturbing learning", is developed to accomplish this task efficiently in a general feedforward net.

2. Formalization of the problem

First one has to model the error function on the previously trained patterns (E , from now on). The problem then becomes the minimization of this function constrained by the pattern equation $P(W)=0$, which expresses that the weights W must be such that the error for the new pattern is zero.

The most complex model we can reasonably aspire to is a second order function without cross-terms in the weights of the network. A linear model is not feasible, as we will see later, and whatever the approach followed, considering cross-terms results in excessive complexity and non local-rules.

It is convenient to consider the actual weights as constants and then put everything as a function of their increments. Our constrained optimization problem can be expressed then as:

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

$$P(\Delta W) = 0$$

Where F is the cost function that estimates the error increment in E , c_i and b_i being constants.

A usual way to approach this problem with neural networks is to add the error function of the new pattern to the F function:

$$\text{Min } \mu F(\Delta W) + \beta P(\Delta W)$$

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

The algorithm we have developed avoids this approximation by converting the unconstrained minimization problem into an unconstrained one, thus tackling the problem in a more direct and efficient manner.

3. Transformation into an unconstrained minimization problem

We begin by introducing a little bit of notation. Let x_j , y_j and f_j be the net input to unit j , the activation and the activation function of unit j . An individual weight will be called w_{ji} (from unit i to unit j) or simply w_i . I_d and O_d denote the input vector and the output vector of the new pattern. I and O stand for the sets of network input and output vectors.

From now on we consider the current weights as constants. Then we define $f^1: \Delta W \longrightarrow H$, H being the hidden units activation vector set, ΔW the weight increment set and f^1 the function which produces the vector of hidden units activations originated by a certain ΔW , given I_d as network input. We define also $f^2: (\Delta W, H) \longrightarrow O$ as the function which returns the net output vector originated by a set of increments and hidden units activations, under I_d as net input. Notice that the function is only defined for $(\Delta W, H)$ such that $f^1(\Delta W) = H$.

Finally our most important function:

$$D: H \longrightarrow \Delta W \quad \text{s.t.} \quad F(D(H)) = \min F(\Delta W)$$

$$f^1(\Delta W) = H$$

$$f^2(\Delta W, H) = O_d$$

$D(H)$ is thus the best solution among the solutions that produce the H hidden activation vector when the new pattern is presented.

The global solution is now:

$$\Delta W^* = D(H^*), \text{ where } H^* \text{ verifies } \min_H F(D(H)) = F(D(H^*))$$

From now on we call $F(H)$ the H dependent function $F(D(H))$.

We have got an unconstrained problem with much less variables than the original one. But the validity of this transformation depends on the fact that D is really a function, and that for each H there is one and only one set of increments satisfying the requirement in the definition. To see this, we start by noting that all the units activations y_i are fixed either by the new input-output pattern or by the H values. Therefore, so are the x_j . In the remaining you must remember that now x_j and y_i are arbitrary constants and the usual equations $x_j = \sum_i w_{ji} y_i$ and $y_j = f_j(\sum_i w_{ji} y_i)$ don't hold but $y_j = f_j(x_j)$ does hold.

Then consider:

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

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

When j runs along all the output and hidden index, the set of R_j are equivalent to the constraints in the definition of D , and $F = \sum_j F_j$. F_j and R_j share all variables and have no one in common with R_k and F_k if $k \neq j$. Then our problem is reduced to independently minimize each F_j under the R_j constraint.

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

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

Now we form G_j : $G_j = F_j + t R_j$. All $\partial G_j / \partial \Delta w_{jk}$ must be zero,

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

$$\Delta w_{jk} = \frac{-b_{jk} - t y_k}{2 c_{jk}} \quad (1)$$

Then using the constraint,

$$\sum_i \Delta w_{ji} y_i = \sum_i \frac{-b_{ji} - t y_i}{2 c_{ji}} y_i = \Delta x_j, \quad \Delta x_j = -\sum_i \frac{b_{ji}}{2 c_{ji}} y_i - \frac{t}{2} \sum_i \frac{y_i^2}{c_{ji}}, \quad t = -2 \frac{\Delta x_j + \sum_i \frac{b_{ji} y_i}{2 c_{ji}}}{\sum_i \frac{y_i^2}{c_{ji}}}, \text{ and substituting in (1)}$$

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

Therefore 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 you use threshold units that take zero as one of their states are used.
- b) All the c_{ji} must be non zero. We have to take care of this when selecting the parameters of the quadratic function (see section 7).
- 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 7).

4. Derivation of the MDL algorithm

Now we can derive a gradient algorithm for our function $F(H)$. We begin by giving F a more explicit form. Developing (2):

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

where
$$E_j = x_j - \sum_i w_{ji} y_i + \sum_i \frac{b_{ji} y_i}{2 c_{ji}} \quad M_j = \sum_i \frac{y_i^2}{c_{ji}}$$

Let D_j be the function that returns the D components corresponding to the incoming connections of unit j ,

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

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

Observe you can take out M_j and E_j from the sumatories

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

so finally
$$F(H) = \sum_j F_j(H) = \sum_j \frac{E_j^2}{M_j} - \frac{1}{4} \sum_{j,i} \frac{b_{ji}^2}{c_{ji}}$$

To reduce greatly complexity it is convenient to work with the parameters w'_{ji} , $w'_{ji} = w_{ji} - b_{ji}/2c_{ji}$. Then $E_j = x_j - \sum_i w'_{ji} y_i$.

So now the important increment to consider is:

$$\Delta_1 w_{ji} = (\Delta w_{ji} + w_{ji}) - w'_{ji} = \Delta w'_{ji} + \frac{b_{ji}}{2c_{ji}} = \frac{E_j y_i}{c_{ji} M_j} = \frac{\left(x_j - \sum_l w'_{jl} y_l\right) y_i}{c_{ji} \sum_l \frac{y_l^2}{c_{jl}}} \quad (3)$$

It is time to find the F gradient with respect to the hidden units activations. Because of the limited activation range, we choose however to calculate $\partial F / \partial x_k$. Note that x_k appears explicitly in F_k but also implicitly in all F_s such that unit s belongs to the outgoing set of units receiving signal directly from unit k ($\text{outc}(k)$).

The gradient is then:

$$\frac{\nabla F}{\nabla x_k} = -2 \frac{E_k}{M_k} + 2 f'_k(x_k) \sum_{s \in \text{outc}(k)} \frac{E_s w'_{sk} M_s + E_s^2 y_k / c_{sk}}{M_s^2} \quad (4)$$

We can construct a gradient algorithm build out of the simplified formulae E_j , $\Delta_1 w_{ji}$ and the above written gradient formula, which have the parameters W' , instead of W . We have only to subtract $b_{ji} / 2c_{ji}$ from all the w_{ji} and then use W' as the current weights. This step is avoided if all b_{ji} are zero.

The gradient formula is easily implemented by an algorithm with a data flow with 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" terms, 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. Actually we cannot update x_j and y_j in a hidden unit before this unit has already contributed to the calculus of the other unit gradients. Because of this independence, the updating order can be anyone, allowing even total overlapping.

Below is the MDL algorithm.

- 0) Fix the input and output pattern I_d and O_d in the network input and output. Choose an initial point for the hidden units activations (for both x_j and y_j).
- 1) $w_{ji} \leftarrow w_{ji} - b_{ji} / 2c_{ji}$
- 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 (4) and update x_j and y_j for all hidden units (backward pass).
- 3) Change weights with (3).

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 this 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.

In this paper we are only investigating the extent to which forgetting can be overcome in distributed representations. For this reason any risky assumption is given up and all experiments are done following true gradient descent until we have a great security of having converged.

5. Implementations details

It is important to realize that weights only change at the end of the search. Each cycle in the loop is slightly more complicated than a back-propagation one. However MDL has the advantage of being fully parallel in

all its stages (except perhaps when choosing the initial point, depending on the method), not only inside a layer but also between layers. Therefore the time cycle would be independent of the number of layers in a truly parallel implementation. In the case of a serial implementation it is also possible to take advantage of several facts. For instance, for a unit in the first hidden layer, M_j and $\sum w_{ji} y_i$ remain constant all along the search. Besides M_j can be shared by all the units receiving signal from the same set of units. For example, if all the consecutive layers in the network are fully connected, M_j can be calculated only once for each layer in each cycle.

Another interesting consideration is that when we fix the network output, the values that should be given are the desired entrance x_j instead of the y_j for the output units. If this is not the case the output units must be able to invert their activation function. In other words, we are saying that the natural desired output data for the MDL algorithm is $f^{-1}(O_d)$. The output units activations are kept for the non learning use of the network, as a sort of postprocessing (an interesting question is whether this is the best way to use the output units also in other learning algorithms).

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. In order to avoid large step sizes producing oscillation and very low rates valid universally, we follow pure gradient descent but with adaptative size step. Our strategy is based on the average slope encountered in the last step, following the gradient direction. This average slope is given by :

$$\text{Average slope} = - \frac{\Delta F}{\sqrt{\sum \Delta H_i^2}} = - \frac{\Delta F}{m \sqrt{\sum \left(\frac{\partial F}{\partial H_i} \right)^2}}$$

$\frac{\Delta F}{F_2} < \text{pendiente min}$
 $\frac{\Delta F}{F_2} < \text{oscillation minimum}$

H_i being hidden activations, and μ the advance rate used in the last step. ΔF is easy to obtain because $F = \sum E_j^2 / M_j$ is always available after the forward pass (we have taken out the constant that is subtracted in F , because it is irrelevant for computing ΔF). It's possible to approximate the average slope suppressing the square root, getting equally satisfying results.

To define a degree of security with which convergence is reached, we specify a positive minimum average slope (in our experiments 0.0005). If the average slope in last step was above this minimum, we consider worthwhile to continue the search multiplying previously μ by a number slightly greater than 1 (1.3). If the average slope was under this minimum, we consider two cases determined by another little small negative parameter that we call maximum allowed oscillation (-10^{-5}), such that if the average slope falls between it and the minimum slope, we consider the search finished. In the remaining case, that where the slope is under the allowed oscillation, a significative increment of the cost function has taken place, thus the step is partially undone, the advance rate reduced and then the step is redone until the average slope is greater than the maximum oscillation allowed. This is implemented through the following sequence of instructions: $H_i \leftarrow H_i - (1-x) \Delta H_i$; $\Delta H_i \leftarrow \Delta H_i x$; $\mu \leftarrow \mu x$; Forward pass. Here x , $0 < x < 1$ is the rate of reduction for μ . We have always taken $x = .5$. We need the maximum allowed oscillation parameter because convergence can only be attained if μ tends to zero and, for each μ there is a certain oscillation around the minimum. This oscillation around the minimum in the MDL space can be lowered by several orders of magnitude only dividing μ by two.

6. Experimental results

We have studied the cost landscapes in a great variety of situations. Negative c_{ji} produced very often landscapes without global minimum and were discarded. With positive c_{ji} , most of the variability in the landscapes features comes from net topology and weight magnitude. This variability manifests itself in the number of local minima and in the largest acceleration rate that can be used without oscillation.

For the unconstrained space, the most complicated cost landscapes were found in very unbalanced

architectures (like 15-2-15), but problems with local minima even in this case arise scarcely if weight magnitude is moderate. If this is not the case, the number of minima grows, specially if the hidden units activations are saturated when activity is normally propagated through the network.

The weight configurations that result from propagating activity through the network normally proved to be good starting points as did those with all the units activations in the middle of the activation range. Intermediate points between these were also safe. With these starting points, and for all the networks in the experiments presented here, we almost never got trapped in local minima.

In all the experiments and for all networks we used the same initial $\mu = 2$. The search with our strong convergence criterium typically includes from 4 to 8 cycles and one or two forward passes alone (when partially undoing a step), indicating the great simplicity of the unconstrained search space. As for the scaling properties, the number of cycles seems to grow slightly with the net size. The growth is steeper with the number of layers.

7. Choosing b_{ji} and c_{ji}

The most obvious election for our constants is that yielded by an instantaneous second order approximation of the error function E . Then $c_{ji} = \partial^2 E / \partial^2 w_{ji}$ and $b_{ji} = \partial E / \partial w_{ji}$. Because as we said it is necessary to have positive c_{ji} , we should take absolute values of the second derivatives. This estimation of E is exactly the one used by Becker and Le Cun (1989) in their Quasi Newton Algorithm and in fact the first step in the MDL algorithm is exactly one step in their second order algorithm. Consequently with this estimation we should expect all the difficulties encountered by them. This leads us to consider the convenience of using the b_{ji} parameters. The MDL algorithm makes this quasi Newton step because it displaces the initial search point to where the minimum of the quadratic function is placed. Then MDL can realize its search without taking into account the b_{ji} (as if they were zero), because at this point first derivatives are zero. Nevertheless this is not true for the real E function unless we are truly in the vicinity of a minimum. We doubtfully can expect always this, and the new point besides not being a minimum can have different second derivatives of that calculated in the starting point. So, we don't make use of $b_{ji} \neq 0$, in order to avoid a step full of troubles that promises little award. There is however an important lesson to be extracted from the above reasoning: you must wait as far as possible to the completion of learning of the previous patterns (by second-order, standard back-propagation or whatever means) before introducing the new patterns. If your context is a situation in which the lags among patterns are known, you must buffer the new patterns according to this rule.

Without b_{ji} and with c_{ji} as second derivatives in the minimum of E , MDL minimizes the same function (constrained by the new pattern) that Le Cun et al. (1990) in their pruning procedure. In a certain sense our technique can be seen as opposed to pruning. Pruning detects the less profited weights to eliminate them. Instead, MDL uses them to introduce new information. For this reason it can be used to compel a network to admit information it can hardly learn (removing local-minima). This suggests alternatives to determine our c_{ji} constants. Pruning methods usually have some measure of weight sensitivity of the error function that could go beyond the strict locality of second derivatives.

The pruning method presented in [Karnin, 1990] uses an easily computable measure, that we have adapted to test as c_{ji} :

$$S_{ji} = \text{abs} \left(\frac{\sum_t \frac{\partial E}{\partial w_{ji}}(t) \Delta w_{ji}(t)}{w^f - w^i} \right) \quad (5)$$

where w^f and w^i are the final weight and the initial weights respectively.

You can use a coarser estimation of damage to the net by eliminating also the c_{ji} parameters. The algorithm is somewhat simpler and permits either saving the cost of calculating c_{ji} (though second derivatives for example are relatively cheap) or working when they are not available. What MDL is calculating in this case is the nearest solution for the new pattern in the weight space. Under total uncertainty in the form 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 did little injury in a study performed by Hinton an Sejnowsky (1986). The study was done with a Boltzmann machine but we think results are valid also for other networks with distributed representations.

Figure 1 presents results for three different networks trained with two different number of patterns (second column of the table). It shows the average error increment in the output units for the previously trained patterns and the average distance covered in the weight space when MDL is applied with diferent c_{ji} values. S_{ji} stands for the sensitivity parameter calculated in (5). We have also tested with $1 + |S_{ji}|$ and $|\partial^2 E / \partial^2 w_{ji}| + 1$, to see how risky are this kind or constants. Net structure is expresed in the obvious way (net 8-4-1 has 8 input units, 4 hidden units and one output unit). Both the previous learned patterns and the new ones were generated randomly from a uniform distribution in $[-1.3, 1.3]$. A symmetric sigmoid was used for the hidden units activation. The table shows averages over a hundred new patterns. The version with the second derivatives is clearly the best even if it moves 20 times the distance covered by the coarsest version. When the network has been trained with a lot of patterns, the advantage of the second version over the first is lowered and the distances covered in both versions are more similar, indicating less variance in the second derivatives and the absence of privileged directions.

Network	Nr. Patt.	Average Error Increment					Average Distance				
		v1	v2	v3	v4	v5	v1	v2	v3	v4	v5
8-4-1	3	0.064	0.032	0.044	0.134	0.059	0.154	0.257	0.173	0.435	0.154
	6	0.063	0.038	0.041	0.134	0.060	0.113	1.092	0.139	0.289	0.116
32-16-1	5	0.025	0.013	0.018	0.013	0.023	0.075	0.554	0.084	0.193	0.075
	20	0.032	0.025	0.026	0.085	0.029	0.091	1.284	0.128	3.680	0.239
32-5-32	5	0.087	0.078	0.064	0.092	0.090	2.264	47.43	2.988	4.080	3.249
	20	0.036	0.034	0.034	0.217	0.217	2.513	2.552	2.562	649.7	649.5

Fig 1. Comparison among different versions of MDL. v1 stands for $c_{ji} = 1$, v2 for $c_{ji} = |\partial^2 E / \partial^2 w_{ji}|$, v3 for $c_{ji} = |\partial^2 E / \partial^2 w_{ji}| + 1$, v4 for $c_{ji} = |S_{ji}|$ and v5 for $c_{ji} = |S_{ji}| + 1$.

8. Revisiting Back-propagation

For comparison purposes we have studied what is the best back-propagation can do with respect to catastrophic forgetting. One of the main results we have obtained is the dependence of back-propagation forgetting on the learning rate. The right way to see the question is this: Back-propagation does not take into account the damage inflicted to the net. However one can expect, with a certain probability, to find one of the nearest solutions following the downhill path, thus approximating the coarsest version of MDL with $c_{ji} = 1$. The problem is that one should not deviate too much from the true gradient line. This requires the use of very small learning rates. The reason for this is that the solutions for the new pattern pervade the weight space. If one takes a too big step, one will be placed in another point crossed by a gradient line that surely brings to a different and farther solution.

We have devised a technique to constrain back-propagation to take only steps which produce error decrements that can be predicted by a linear aproximation of the error function. We took the criterium:

$$\text{abs} \left(1 - \frac{\sum_{j,i} \Delta w_{ji}(t) \frac{\partial E}{\partial w_{ji}}(t)}{E(t+1) - E(t)} \right) < \text{exigence}$$

to decide whether the learning rate $\mu(t)$ that produced $\Delta w_{ji}(t)$ was too big. If all the steps along the learning process respect this criterium, we can say that the trajectory followed by the algorithm approximates the trajectory that would follow back-propagation as a continuous dynamical system. The fidelity to the continuous path will be controlled by the exigence parameter. In this way we can define back-propagation time as $\sum \mu(t)$, an estimation of the time the continuous system would need to complete the trajectory. We used back-propagation time as a reference measure with which to compare objectively recovery times, which does not depend on any parameter as back-propagation and its versions do. An equivalent measure would be $\mu \cdot nc$, nc being the number of back-propagation cycles with a fix learning rate μ , such that the number of cycles with learning rate $\mu / 2$ would be approximately doubled.

Fig 2a

Our technique always respects the above criterium trying to maintain the biggest $\mu(t)$ possible at each t . Details are a bit complicated because we had to deal with precision errors, and will be presented elsewhere. It can be thought that the same linearizing technique used to find back-propagation time would alleviate back-propagation problems with forgetting, by adjusting trajectory to the gradient line, while preserving however a certain efficiency. We tested the idea and although it does not lead to improvement in terms of learning iterations, when a fix μ that arrives to a solution just as far is chosen, it at least avoids an important problem: that of finding a reasonably big μ that produces acceptable results.

In fact this is not the most appropriate approach to keep trajectory near the gradient line. It would be more useful to control the angle change between two steps. But, in addition to the difficulties encountered in calculating back-propagation time, an algorithm for this purpose implies others, like the need of waiting two iterations before knowing the correctness of the first.

In Figure 2a the net 32-16-1, trained with 20 random patterns, undergoes the introduction of a new pattern with different back-propagation learning rates. The distance from the starting point to the final one in the

Fig 2b

weight space and the number of cycles needed are shown. The pattern is considered to have been learned when the average absolute error in the output unit is 0.01. Although it is not represented in the graphic, applying the linear approximation technique to introduce the pattern (exigence = 0.1) needed 44 cycles, moving a distance of 267. Figure 2b indicates how the distances to the starting point showed in 2a 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. Comparison with the coarsest version of MDL (c_{ji}