# Recurrent Backpropagation and Hopfield Networks

Luis B. Almeida and João P. Neto
INESC
Apartado 10105
P-1017 Lisboa Codex
Portugal

This paper has two parts. In the first one, an intuitively simple proof of the extension of backpropagation to recurrent networks is given. In the second part, preliminary results on the application of recurrent backpropagation to the training of Hopfield networks are presented.

## 1 - Introduction

Backpropagation is a well known learning technique for multilayer perceptrons. As originally introduced, it was applicable only to feedforward networks, i.e. networks with no recurrent connections [1]. The extension of this rule to recurrent networks was first developed by this author [2]. Pineda [3] independently derived the same result. In this paper, a new form of the derivation is first presented, which is easier to grasp from an intuitive viewpoint. Then, some preliminary results on the training of Hopfield networks through recurrent backpropagation are described. These results suggest that backpropagation is a viable alternative for the training of such networks, possibly yielding some advantages, like the use of hidden units, and the training of analog-valued stable patterns. The paper is organized as follows: Section 2 presents the new derivation of recurrent backpropagation, and section 3 gives the results on training of Hopfield networks. Section 4 concludes.

## 2 - Recurrent backpropagation

Backpropagation is a gradient optimization technique for minimizing the total quadratic error of the outputs of multilayer perceptrons. As originally developed for feedforward networks [1], it involves a backward propagation of errors which can be viewed as a propagation through an *error propagation network*. As shown in [2], this network can be obtained from the multilayer perceptron by the application of two successive operations: linearization and transposition [4]. In [2], it was mathematically shown that this rule generalizes to recurrent perceptrons, i.e. that gradient optimization can be performed in the same way in these networks, the error propagation network being still obtained through linearization and transposition of the perceptron. Next, we shall give a more intuitive version of this proof.

NATO ASI Series, Vol. F 68
Neurocomputing
Edited by F. Fogelman Soulié and J. Hérault

As a step towards the derivation of the gradient learning procedure, we shall first find a way to compute the partial derivative of an output relative to a weight. Consider a general nonlinear network **P** with an output $o$ and a linear branch of gain $a$, as depicted in figure 1-a. The external inputs of the network are to be kept fixed during the partial differentiation, and therefore they can be considered to be contained within the network itself, without loss of generality. This is why they are not shown in the figure. The network is assumed to be at a fixed point.

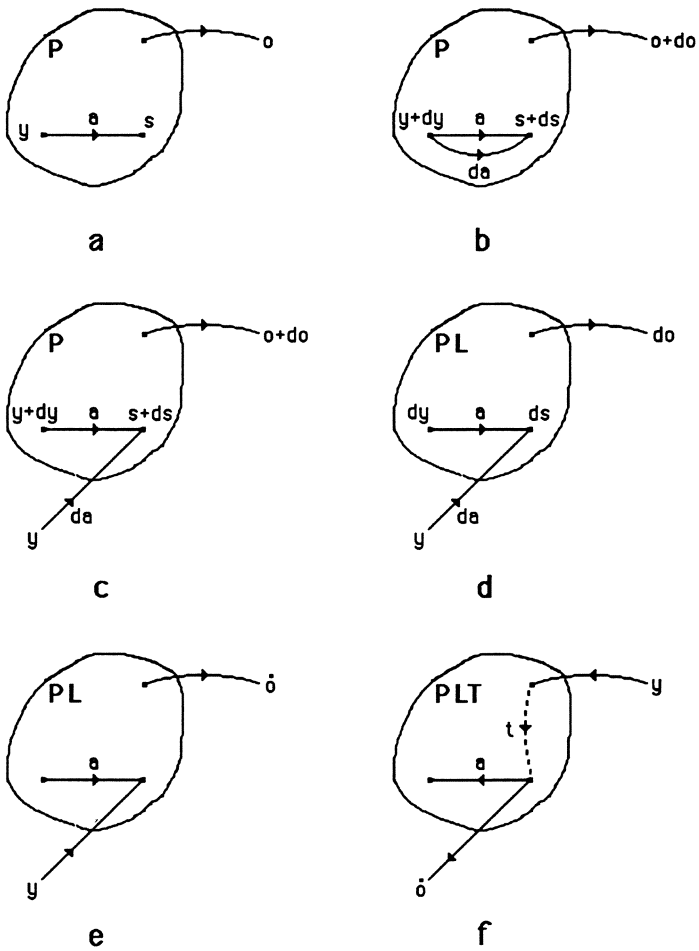


Figure 1 - Computation of the partial derivative of the output of a nonlinear network, relative to a branch weight. See text for explanation.

To compute the partial derivative, we shall give an infinitesimal increment to the branch gain. This is equivalent to adding an extra branch with gain *da* (figure 1-b). Since the network can have feedback connections, there will be increments in all node variables, including the node at the input to the branch under consideration. The net output of the new branch will be *(y+dy).da*, or simply *y.da*, if we discard the higher order term. The same result can be obtained by using and external input with value *y*, through a branch with gain *da* (figure 1-c). If we now linearize around the original fixed point, considering only increments, we will obtain the network of figure 1-d, which we have designated by **PL**.

Network **PL** is linear, and therefore if we divide its net input by *da*, the output will be divided by the same amount. Dividing the net input by *da* can be accomplished by changing the gain of the input branch to unity, as shown in figure 1-e. The output will now be $\partial o/\partial a$, which we shall designate by *o*, for compactness.

The network of figure 1-e is linear, and has a single input and a single output. Therefore, the transposition theorem [4] can be applied to it, yielding the transposed network (**PLT**) which, when its input is *y*, still produces the output *o* (figure 1-f). If we call *t* the gain of this network from input to output, we can write

$$o = y\,t$$

We shall now use this expression of the partial derivative to obtain the gradient learning rule. Consider a perceptron (figure 2-a) with a fixed input pattern. We can write for each output $o_p$ (figure 2-b)

$$o_p = \frac{\partial o_p}{\partial a_{ij}} = y_i\ t_{pj} \qquad (1)$$

If the perceptron has several outputs, and $O$ is the set of indexes of the units that produce external outputs, the squared error for the given input pattern is

$$E = \sum_{p \in O} e_p^2$$

where

$$e_p = d_p - o_p$$

is the error of output *p*, $o_p$ and $d_p$ being the output and the desired value, respectively, for the given input pattern. Now,

$$E = \frac{\partial E}{\partial a_{ij}} = -2 \sum_{p \in O} e_p\ o_p$$

But, using eq. (1)

$$\dot{E} = -2 \sum_{p \in O} e_p \; y_i \; t_{pj}$$

or

$$\dot{E} = -2 \; y_i \sum_{p \in O} e_p \; t_{pj}$$

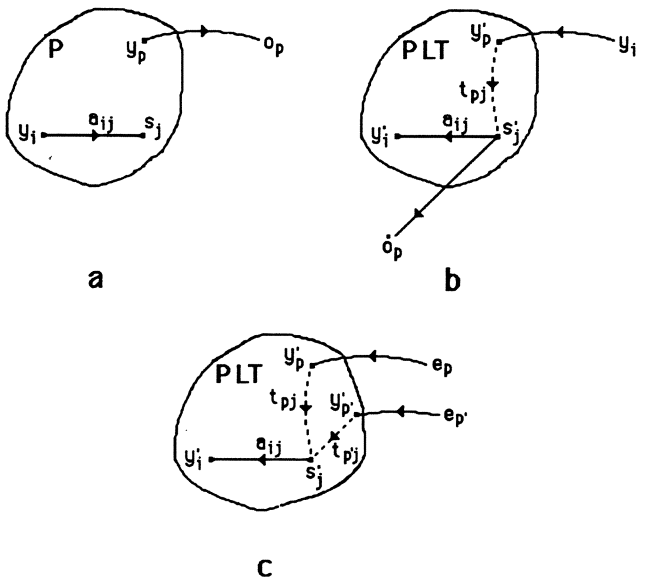


Figure 2 - Computation of the partial derivative in the case of a recurrent perceptron. See text for explanation.

Refering to figure 2-c, and taking into account that network **PLT** is linear, we can finally conclude that:

$$\dot{E} = -2 \; y_i \; s'_j$$

where $s'_j$ is the value obtained at the corresponding node when the output errors are all simultaneously applied to the respective inputs of the transposed network, as shown in that figure. This is a direct extension of the backpropagation rule of feedforward networks: the

derivatives are still obtained by applying the output errors to an error propagation network, which is obtained by linearizing and transposing the original perceptron.

## 3 - Training of Hopfield networks

Hopfield networks [5,6] have been the centre of much interest from researchers. Training methods include Hopfield's original storage prescription [5], pseudo-inverse methods [7] and extensions of the perceptron learning procedure [8]. In this section, we present preliminary results on an investigation of the possibility of training Hopfield networks through recurrent backpropagation.

A fully connected perceptron with no inputs, and with symmetrical weights, i.e., in which $a_{ij}=a_{ji}$ for all $i,j$ is equivalent to a graded Hopfield network [6] if the same dynamical behavior is assumed for the units as was done by Hopfield. The dynamical evolution of such a network can be characterized by an *energy function.* The network evolves in such a way that the energy function always decreases. The points where the network stabilizes are points of stationarity (usually local minima) of this function. In Hopfield networks, the locations of these fixed points are of special interest, since they are used to store patterns. As we have seen, backpropagation can be used to train a network with feedback, moving its fixed points towards desired positions. Therefore, we can conjecture that backpropagation can be used to store a desired set of patterns in a Hopfield network. In the next sections, results obtained with this kind of training, are described. Before presenting these results, however, a point about network initialization should be discussed. The backpropagation procedure can only train fixed points, it cannot train the dynamical behavior of the network. Consider two patterns to be stored, and assume that with its initial weights, the network would evolve from both patterns to the same fixed point. Backpropagation training cannot be expected to split this fixed point into the two that we would desire the network to have. Therefore, steps should be taken to ensure that, initially, each input pattern evolves to a different fixed point.

### 2.1- Training of fixed points

A 10-unit network was used for this test. The network was fully connected, including connections from each unit to itself (this is slightly different from Hopfield's original topology, but is similar to what has been used by other authors [7]). The patterns to be trained were randomly generated vectors of 10 components each. Each vector component could take the values - 0.8 and + 0.8, with equal probability. The sigmoids used in the network's units had an output ranging from -1 to +1.

Training was performed as follows. As an initialization procedure, the network was first trained with recurrent connections opened, i.e., in a feedforward configuration (see figure 3 for an example). In this mode, it was trained to perform an identity mapping on the training patterns. This training was performed for a number of iterations sufficient to ensure that each training pattern would evolve to a different fixed point, when recurrent connections were closed.

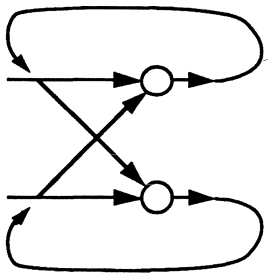

Figure 3 - Illustration of the weight initialization procedure in a 2-unit network.

After this initialization, recurrent connections were closed. Each pattern **p** to be trained was clamped on the network, which was then released, evolving to some fixed point **f**. This fixed point was trained towards the originally clamped pattern, i.e. the output error vector was computed as $\mathbf{e} = \mathbf{p} - \mathbf{f}$, and this error vector was input to the error propagation network in the usual way, for gradient computation. Weight updates were performed after each sweep through the whole set of training patterns.

Table I shows the results obtained for various training set sizes. The statistics were collected after training, by systematically testing the 1024 possible input patterns. Each one was clamped onto the net, which was then released, evolving to some fixed point. The pattern was considered to be stable (and therefore stored) if the signs of the unit outputs at the fixed point were the same as those of the corresponding components of the initially clamped pattern. Spurious stored patterns were those stable ones that did not belong to the training set. The table shows a performance close to what could be expected [9]: the number of spurious patterns is very low initially, and increases steeply when the training set size is of the order of the network size (note, however, that we exert no direct control over spurious states, and that some form of *unlearning* [10] might reduce their number, improving the network's performance).

| trained | spurious | trained | spurious |
|---|---|---|---|
| 6 | 0 | 15 | 53 |
| 8 | 1 | 20 | 145 |
| 9 | 9 | 30 | 324 |
| 10 | 14 | 40 | 745 |
| 12 | 18 | 50 | 917 |

Table I - Results of training of random patterns; **trained** - training set size; **spurious** - number of spurious stable patterns.

3.2 - Training of regions of attraction

For a better control of the network's performance, one may want to train not only the fixed points, but also each fixed point's region of attraction (in [8], another procedure for training basins of attraction is given). For this purpose, we used the following procedure: We randomly selected a pattern **p**. Assume that this pattern was desired to belong to the region of attraction of some stable pattern **q**. After clamping **p** and releasing the network, the resulting fixed point **f** was trained to move towards the state **q**. Then, a new pattern **p** was randomly selected, and the procedure repeated. Weight update was performed after each pattern presentation. As network initialization we used a very simple deterministic method: we initialized each "self" weight $a_{ii}$ to a relatively large value (typically 3), and each $a_{ij}$ (with $i \neq j$) to zero. In this way, each unit would initially be bistable, and independent of other units, i.e., all possible binary patterns would be stable (note that this initialization could not be used in the case of section 3.1, since there we were training starting only from the desired fixed points, and therefore training would never modify the network's behavior if these patterns were already stable).

Small 2- and 3-unit networks were used in these tests. Figures 4 and 5 depict some of the behaviors that we were able to train, showing that a good control over the regions of attraction can be obtained by this procedure, at least for these small networks.

Figure 4 - Basins of attraction trained on a 2-unit network.

Figure 6 shows an interesting case. This behavior, which was trained on a network of four units (three visible and one hidden), corresponds to the exclusive-or case. It cannot be obtained without hidden units, and therefore it cannot be obtained with any of the other available training methods. The network was operated in the following way. Each pattern was kept clamped onto the visible units long enough for the hidden unit to stabilize. After that, the visible units were unclamped, and the network was allowed to relax to some fixed point. Weight initialization was performed as outlined above, for the visible units. Weights between visible units and the hidden unit were randomly initialized, and that unit's feedback weight was initially set to zero. The remainder of the training procedure was as described above in this section.

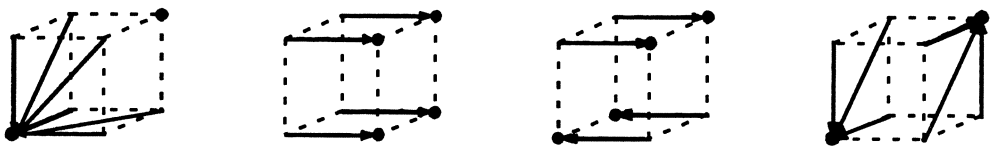

Figure 5 - Basins of attraction trained on a 3-unit network

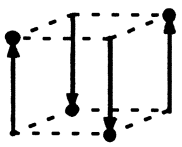

Figure 6 - Exclusive-or behavior trained on a net with 4 units (3 visible and 1 hidden)

3.3 - Discussion

These preliminary results suggest that recurrent backpropagation is a viable technique for training graded Hopfield networks. We have shown that it is possible to train both stable patterns and regions of attraction, and we have given an example of the training of a network with one hidden unit. However, these results also suggest that a better control over the intialization and learning is needed. As an example, a behavior which was hard to train, requiring very precise adjustment of the training of parameteres, and whose success depended on the random ordering of patterns during training, is shown in figure 7-a. Most often, after some training time, the fixed points corresponding to both right-hand patterns would merge,

and then the single resulting fixed point would be trained to the left 50% of the time, and to the right another 50%, finally stopping midway, as shown in figure 7-b.

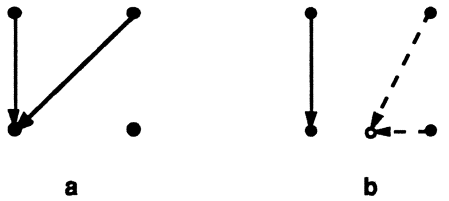


Figure 7 - Example of a problem in the training of basins of attraction; **a** - desired behavior; **b** - behavior that was most frequently obtained.

An interesting application of backpropagation would be to train a graded Hopfield network to have given analog-valued stable patterns. An example of two patterns that were successfully stored in a 10-unit network, using the procedures described in section 3.2, is given in figure 8. However, there is still very little experience on this kind of application.

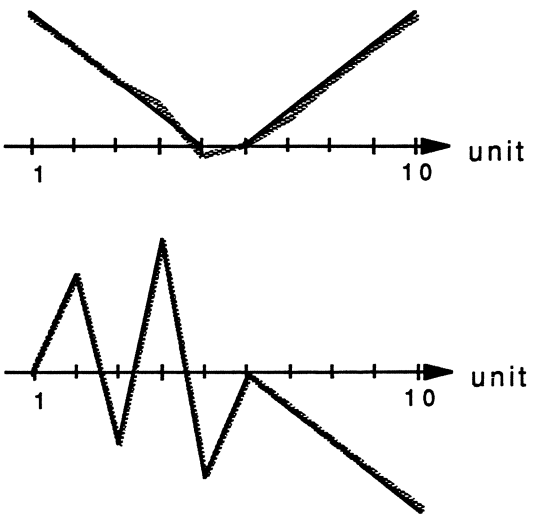


Figure 8 - Two analog patterns that were trained on a 10-unit network. Horizontal axis - unit number. Vertical axis - unit output. The unit outputs have been joined by straight lines for better visibility. Thin black lines - trained pattern. Thick gray lines - recalled pattern.

## 4 - Conclusions

We have given an intuitively simple proof of the extension of the backpropagation learning rule to recurrent networks. We have also presented results on the application of this rule to graded Hopfield networks, both for the training of stable patterns and of regions of attraction. These results lead us to think that recurrent backpropagation is an alternative method for the training of these networks. Two of the potentially interesting applications of this form of learning, would be the training of Hopfield networks with hidden units, and the storage of analog-valued patterns. Though one example has been presented of each of these cases, further work is needed to assess their general feasibility.

## References


1. D. Rumelhart, J. McClelland and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge, MA: MIT Press, 1986.
2. L. Almeida, "A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment", *Proceedings of the 1987 IEEE First Annual International Conference on Neural Networks*, S. Diego, CA, June 1987.
3. F. Pineda, "Generalization of Backpropagation to Recurrent and Higher Order Networks", *Neural Information Processing Systems*, D. Anderson (ed.), American Institute of Physics, 1988.
4. A. Oppenheim and R. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1975.
5. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons", *Proceedings of the National Academy of Sciences of the USA*, Vol. 81, pp. 3088-3092, May 1984.
6. J. Hopfield and D. Tank, "Neural Computation of Decisions in Optimization Problems", *Biological Cybernetics*, Vol. 52, pp. 141-152, 1985.
7. L. Personnaz, I. Guyon and G. Dreyfus, Information Storage and Retrieval in Spin-Glass Like Neural Networks", *J. Physique Lett.*, **46**, pp. L-359 - L-365, 1985.
8. E. Gardner, N. Stroud and D. Wallace, "Training with Noise, and the Storage of Correlated Patterns in a Neural Network Model", Edinburgh Preprint 87/394, University of Edinburgh, 1987.
9. E. Gardner and B. Derrida, "Optimal Storage Properties of Neural Network Models", *J. Phys. A: Math. Gen.*, **21**, pp. 271-284, 1988.
10. J. Hopfield, D. Feinstein and R. Palmer, "Unlearning has a Stabilizing Effect in Collective Memories", *Nature*, vol. 304, pp. 158-159, 1983.