Neural networks with periodic and monotonic activation functions: a comparative study in classification problems

Enrique Romero *
Departament de LSI
Universitat Politècnica de Catalunya
eromero@lsi.upc.es

Josep M. Sopena
Lab. de Neurocomputació
Universitat de Barcelona
pep@axon.psi.ub.es

René Alquézar *
Departament de LSI
Universitat Politècnica de Catalunya
alquezar@lsi.upc.es

Joan L. Moliner
Lab. de Neurocomputació
Universitat de Barcelona
joan@axon.psi.ub.es

February 14, 2000

Abstract

This article discusses a number of reasons why the use of non-monotonic functions as activation functions can lead to a marked improvement in the performance of a neural network. Using a wide range of benchmarks we show that a multilayer feed-forward network using sine activation functions (and an appropriate choice of initial parameters) learns much faster than one incorporating sigmoid functions - as much as 150-500 times faster - when both types are trained with backpropagation. Learning speed also compares favorably with speeds reported using modified versions of the backpropagation algorithm. In addition, computational and generalization capacity increases.

*This work was supported by Consejo Interministerial de Ciencia y Tecnología (CICYT), under project TAP1999-0747

1 Introduction

The term "classification" covers a wide range of problems in Artificial Intelligence. In general, given a finite number of variables which are assumed to characterize the problem in question, and a finite number of classes, a classification system assigns a particular class to each set of values of the variables.

With this definition, classification can be understood as the application of a function with a domain in the space of variables and image in the space of classes. A classification system is simply a calculator of this function. In many real problems, these functions are unknown, and they must be approximated from a set of given points. The problem of approximating functions can be defined as follows: given an unknown function $f(\vec{x})$, defined on an interval I of a space of finite dimension, whose value is only known in a set of points $X \subset I$, find a function $g(\vec{x})$ such that the values of the two functions at the points of interval I are as similar as possible.

Neural networks offer a method for approximating functions. A feed-forward network with a single hidden layer and an output neuron with linear activation function approximates a function as follows:

$$g_j(\vec{x}) = bias_{oj} + \sum_{i=1}^{nhid} \alpha_{ij}\theta(\vec{x}\vec{\omega}_i + bias_i) \quad j = 1 \dots nout$$

where $nout$ is the number of neurons in the output layer, $nhid$ is the number of neurons in the hidden layer, α_{ij} are the weights that connect the neuron i of the hidden layer to the neuron j of the output layer, $\vec{\omega}_i$ is the weights vector that connects the input layer to the neuron i of the hidden layer and θ is the so-called activation function. The biases are external parameters for each neuron, and can be considered as weights with a simple transformation.

Goodness of fit is calculated on the basis of a measurement of the approximation error, known as the error function (E). The quadratic error function is the most usual:

$$E = \sum_{\vec{x} \in X} (f(\vec{x}) - g(\vec{x}))^2$$

in which $f(\vec{x})$ is named target, $g(\vec{x})$ output and X the training set. The error is a function of the free parameters in the system. In the case of neural networks it is a function of the weights of the connections, and can be represented as a multidimensional surface in a space in which the free parameters constitute the coordinates. In a feed-forward network with a hidden layer, a surface of these characteristics may have several minima, some of which are local, and others global. The optimal fit is obtained at a global minimum of this surface.

A learning algorithm aims to lead the system from an initial state (in which the values of the parameters are randomly assigned) to a global minimum of error. Once this is achieved, the network's generalization capacity is measured by its ability to calculate correctly the values of the function at points that do not belong to the training set. The most frequently used learning algorithm is backpropagation (BP). BP determines the direction in which we have to move in the space of weights (i.e. how we have to modify the weights) calculating the gradient vector in this space for the points of the training set. This vector points in the direction in which the rate of increase of E is greater. The modification of the weights is calculated as follows:

$$\begin{aligned} \Delta\omega_{ij}(t) &= -\eta\frac{\partial E}{\partial w_{ij}} + \alpha\Delta\omega_{ij}(t-1) \\ \omega_{ij}(t+1) &= \omega_{ij}(t) + \Delta\omega_{ij}(t) \end{aligned}$$

where η is a positive number named learning rate and α a positive number named momentum. This last parameter is introduced to stabilize the learning process.

A good learning algorithm must have the following properties (among others):

- Fast learning speed.
- Computational capacity (approximation to complex functions).
- Generalization capacity.

A number of attempts to improve these three properties have been made since BP was first proposed. [Fahlman & Labiere 1990] states that BP's lack of speed is one of the main reasons why neural networks have not been more widely used. In addition, although in theory a neural network can approximate any function [Cybenko 1989], in practice finding the right set of parameters may be extremely difficult. Finally, in certain problems, even if the training set is learnt, the generalization is unsatisfactory. Several modifications of BP have been proposed in an attempt to improve some of these properties:

"Delta-Bar-Delta Rule" [Jacobs 1988], "Quickprop" [Fahlman 1988], "Supersab" [Tollenare 1990], "Cascade Correlation" [Fahlman & Labiere 1990], "RPROP" [Riedmiller & Braun 1993], etc.

None of these proposals have considered modifying the activation functions. The most frequently used activation functions are monotonic, such as the sigmoid function and the hyperbolic tangent. Recent years have seen the introduction of Radial Basis Functions Networks (RBFN), which use non-monotonic functions such as Gaussian functions. Their performance with a wide range of problems does not differ greatly from that of sigmoid functions, although in certain problems it is superior [Wettschereck & Dietterich 1992].

This study compared the behavior of multilayer feed-forward networks (multilayer perceptrons) with periodic activation functions and multilayer feed-forward networks with monotonic activation functions. In our experiments, the sine was the periodic activation function. A wide range of benchmarks indicated that the behavior of periodic functions was clearly superior, in terms of learning speed, computational capacity and generalization.

2 Periodic functions vs. monotonic activation functions

[Morita *et al.* 1990] demonstrated empirically that the storage capacity of an associative memory with non-monotonic activation functions can be several times greater than the capacity of the same architecture with monotonic activation functions. Later, [Yoshizawa *et al.* 1993] demonstrated this formally. One might think - although this has not been formally proven - that the same property may hold for multilayer perceptrons. As multilayer perceptrons are highly non-linear, demonstrations of this type are difficult to obtain; for example, there is as yet no theorem of convergence for BP. However, there are a number of reasons for thinking that the choice of non-monotonic functions as activation functions - in particular, sine functions - can produce a notable improvement in the performance of a neural network.

A classifier acts by dividing the space of variables or input space into regions, so that in each region there are only representatives of one class. In the case of a neural network, each unit contributes to dividing the input space in a particular way. It is well known that units with monotonic activation functions generate a single hyperplane which divides the input space into two regions. The combination of several of these hyperplanes can give rise to complex classifications [Lippmann 1987]. A unit with a non-monotonic activation function can divide the space into more than one hy-

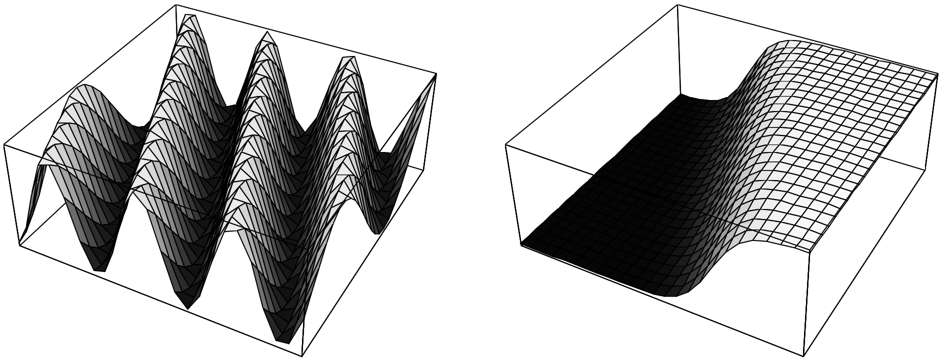

Figure 1: Comparison between the sine and sigmoid functions in a two-dimensional space for the same argument ($\vec{\omega}\vec{x} = 5x + 2y$).

perplane. If the function is periodic, the number of hyperplanes is infinite, and it can be interpreted as a wave front spreading through the space of variables [Monte & Sopena 1993].

For a clearer idea of how sine functions behave, let us imagine a two-dimensional space such as the one in figure 1. In this space a function $sin\,(\vec{\omega}\vec{x} + bias)$ represents a sinusoidal surface whose contours of equal height are given by $\vec{\omega}\vec{x} = constant$. The separation between two consecutive contours is related to the wavelength $2\pi/\sqrt{\vec{\omega}^2}$. The orientation of these contours is given by $\arctan(\omega_1/\omega_2)$. The greater the weights, the lesser the distance between two levels of the same height, and therefore the input space will be divided into smaller regions.

Let us imagine a feed-forward network with a single layer of hidden units with sine activation functions and a linear output unit. This network will approximate a function by effecting a superposition of sinusoidal functions:

$$g\,(\vec{x}) = bias_o + \sum_{i=1}^{nhid} \alpha_i \sin\,(\vec{x}\vec{\omega}_i + bias_i)$$

We will have as many sinusoidal surfaces as there are units in the hidden layer. The characteristics of each of these surfaces are given by the weights of the connections that the corresponding hidden units receive. We will call these weights non-linear parameters ($\vec{\omega}_i$). These different sinusoidal surfaces combine linearly to produce the output. The parameters that adjust this linear combination are the weights of the connections between the hidden units and the output units. We will call these parameters linear parameters (α_i).

Intuitively, it appears that by combining several of these units with periodic functions we should obtain more capacity to divide the space into differentiated regions than with sigmoidal functions (see figure 1); it should therefore be possible to approximate more complex functions. This approach also seems to simplify the treatment of problems with regularities or with some type of periodicity, which are extremely difficult to solve using monotonic functions.

However, networks with sine functions present a problem [Sopena *et al.* 1993]. The error function is convex as regards the linear parameters, but is clearly non-convex for non-linear parameters. There are a large number of local minima. Networks with sigmoid functions can have a large number of local minima, although probably less than networks with sine functions. This is because the function is non-monotonic, as in the case of the RBFN. An algorithm based on the correction of error, such as BP, can easily be trapped in a local minimum before it reaches a global minimum. Consequently, with non-monotonic activation functions it is difficult to make the weights vary sufficiently from their initial position. A solution proposed for the RBFN [Lowe 1991] is to assign an initial set of weights that is appropriate for the problem in question. In this way the region of the parameter space to be explored is delimited. With appropriate initial assignations, the probability of falling in a local minimum is substantially diminished.

Another important factor is the different time scales in which the network's parameters evolve. Non-linear parameters are difficult to adjust, and evolve at a much slower pace than linear parameters, which adjust much more easily [Lowe 1991]. This means that, in order to obtain satisfactory performance, the initial ranges of weights must be different for each layer (large in the first layer, and small in the second).

In the benchmarks that follow we will check empirically the aspects of the behavior of the networks with sine activation functions. We will show that when the sine function initially has a wide, appropriate range of values of weights its behavior is clearly superior to that of the sigmoid function. We will also show that in general the behavior of the sine function is not very different from that of the sigmoid function if the initial values of the weights are not suitable.

3 Experiments

The benchmarks we used in the comparison have been frequently studied elsewhere. Most of them are taken from the *NN-Bench Collection* of the University of Carnegie Mellon.

3.1 General method

We now describe the training conditions, and comment on some of the criteria used in the experiments.

Each complete presentation of the training set is called an epoch. The learning speed is measured as the number of epochs that the network needs to learn the training set. In the experiments, training is considered complete (the stop criterion) when all the examples have been learnt up to a given tolerance. In some cases a different criterion is used, because learning all the examples is impossible, or even not convenient for generalization purposes. There are two learning modes: either the weights are modified at the end of each epoch (the batch mode) or after the presentation of each example (the pattern mode). The effectiveness of each learning mode depends on the problem to be solved [Hertz *et al.* 1991]. Throughout our experiments we used the pattern method, as this method seems less likely to be trapped in a local minimum [Haykin 1994].

The architecture used was feed-forward with a layer of hidden units in all cases. The best-suited architecture depends on the type of problem, although the most frequently used, and the simplest, is the one with a single hidden layer. The number of units in each layer is indicated by the expression I-H-O, where I, H and O are the number of units in the input, hidden and output layers respectively. Since the functions to be approximated in all the experiments have only two possible values, a sigmoidal activation function in the neurons of the output layer was used in all of them. The weights were initially assigned randomly to a particular interval ($-wrange/2$ and $wrange/2$). As we said above, the value of this range can vary for each layer of weights. This is shown in the results tables as follows: I-H:$wrange$ refers to the weights in the first layer, and H-O:$wrange$ to those of the second. Each experiment was conducted ten times for each of the configurations of initial parameters.

To simplify the initial parameter fit, momentum was set at 0. With this value, adequate performance was achieved in all the problems. Throughout the experiments, bias in the hidden layer was only used in the case of the "Quadrants". If we have no bias, only the sine functions can perform the approximation (the bias represents the phase, and, in particular, converts the sine into cosine when $\pi/2$). Certain functions such as the "Negation N" (see section 3.3) are symmetrical and thus the optimal approximation involves the use of cosines. One way of ensuring that this function can only be approximated with sines is to desymmetrize it by transforming the input. To do so, we assign only positive values to all the variables in the input space. We use this transformation in all cases except the "Two spirals" problem and the "Quadrants".

3.2 Two spirals problem

The "Two spirals" problem has been perhaps the most widely used benchmark in order to test the effectiveness of new learning algorithms, or proposed modifications of BP. It consists in identifying the points of the plane which are on the surface which form two interlocking spirals that go around the origin three times. The training set comprises 194 points, half of which have +1 as target, and the other half -1.

Standard BP with a simple architecture has not found a solution to this problem (see [Fahlman & Labiere 1990]). Weiland, in an unpublished manuscript, using a modification of BP, needed between 150,000 and 200,000 epochs to learn it. Using standard BP with a complex architecture (2-5-5-5-1 with shortcuts, in which each unit receives activation from all the units in previous layers), [Lang & Witbrock 1988] solved it in 20,000 epochs. "Supersab" needed an average of 3,500 epochs, "Quickprop" 8,000, "RPROP" 6,000, and "Cascade Correlation" 1,700.

To solve this problem we constructed a network with architecture 2-16-1, using the sine as the activation function in the hidden layer, and the hyperbolic tangent in the output layer. This architecture is the simplest of those used to date to deal with this problem (i.e. it has the fewest parameters). Results are shown in table 1. The importance of the range of initial weights is clear. With small ranges for the non-linear parameters learning is not possible: for example, with a range of 1 the network always falls in a local minimum after only a few epochs. In these local minima the number of examples learnt does not exceed 52%. With a range of 3, the network continues to fall in a local minimum after a few epochs. This time the number of examples learnt is around 80%.

The supposition that the linear parameters have a much faster temporal evolution than non-linear parameters and should therefore be initialized in a different way is also confirmed by the results shown in table 1. With the correct range, we see that the learning speed obtained with sine functions (and with a simpler architecture) is some 40 times greater than the fastest current method.

wrange	%lm	epochs	generalization
I-H:1 H-O:1	100(100.0)	-	-
I-H:3 H-O:1	100(154.9)	-	-
I-H:7 H-O:1	54(186.8)	46.0	97.6%
I-H:7 H-O:3	85(186.8)	237.5	94.7%
I-H:7 H-O:0.001	28(191.0)	42.27	98.3%

Table 1: Results for the "Two spirals" problem, with different ranges of initial weights and learning rate 0.006. In the column '%lm' the first figure indicates the percentage of failures to learn the training set as regards the total number of executions, and the second shows the average number of examples learnt in these cases. The column "epochs" indicates the average number of learning epochs for succesful runs. The last column shows the average percentage of correct generalization for succesful runs.

The generalization obtained by "Cascade Correlation" can be seen in figure 2a. It is clear from figure 2b that the generalization obtained by a multilayer network with sine functions is better.

3.3 Negation N

In this problem the input patterns have N+1 bits (binary coded). The last bit indicates whether the output is to be a copy of the N first bits of the input or their negation (1-complement).

In [Rumelhart *et al.* 1986], with a 4-3-3 architecture with sigmoids, the network required some 5,000 epochs to learn the function. We constructed the same architecture using the sine as the activation function of the hidden layer and the hyperbolic tangent in the output layer. The results are shown in table 2. This case, like the others, confirms the importance of the ranges of the initial weights.

3.4 Parity N

In this problem the input patterns comprise zeros and ones. The network has to determine whether the number of ones in the input is odd or even.

[Rumelhart *et al.* 1986], using sigmoids, solved the problem of "Parity 4" with a 4-4-1 architecture in 2,825 epochs. [Tesauro & Janssens 1988] solved the problem of "Parity 8" with an 8-16-1 architecture in 2,000 epochs. Table 3 shows the results obtained by the "Cascade Correlation" algorithm. As can be observed, the difficulty of the problem increases with the size of the input.

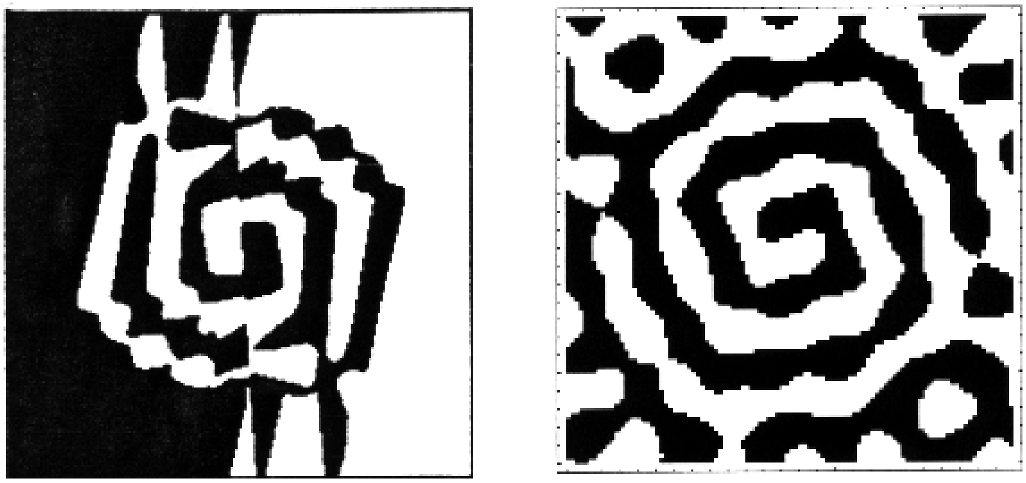

Figure 2: Generalization obtained by [Fahlman and Labiere, 1990] (a) and generalization obtained with sine functions (b).

The "Parity N" function is periodic. Using the sine function learning can be achieved with a perceptron (a perceptron is a feed-forward network without hidden layers) in around 3 epochs, regardless of the value of N (see [Monte & Sopena 1993]). Learning with a multilayer perceptron with sines is slightly more difficult; the average number of epochs is around 6. The most interesting aspect of learning in this function is in the capacity for generalization that the network acquires once it has learnt. The number of patterns for "Parity 10" is 1,024. [Fahlman & Labiere 1990] used "Cascade Correlation" to learn 50% of these patterns and performed a test with the rest, obtaining 96% generalization. When they used 25% of the patterns to train the network, generalization fell to 90%. In our case, we constructed a training set that contained only 5% (selected at random) of the 1,024 examples and a test set of the remaining 95%. With sine functions, either with a perceptron or with a multilayer perceptron, we obtained 100% generalization. These results demonstrate the difficulty that sigmoid type functions face in approximating periodic functions, or functions with a periodic component.

3.5 Separation problem

[Yang & Wanzhen 1993] presented a classification problem that poses extreme difficulties for standard BP. In this problem, as in the "Two spirals" situation, standard BP is trapped in local minima. To avoid this,

wrange	%lm	epochs
I-H:1 H-O:1	43	161.0
I-H:3 H-O:1	34	33.3
I-H:5 H-O:1	9	58.6
I-H:3 H-O:0.05	10	29.3
I-H:5 H-O:0.05	5	30.0

Table 2: Learning results for the "Negation N" problem with different ranges of initial weights and learning rate 0.04. The percentage of local minima ('%lm') and the average number of learning epochs for succesful runs is shown.

N	2	3	4	5	6	7	8
epochs	24	32	66	142	161	292	357

Table 3: Results of "Cascade Correlation" for "Parity N".

[Yang & Wanzhen 1993] apply a method (BP with homotopy) that reaches a solution after 5,172 epochs. With the same architecture as in [Yang & Wanzhen 1993] (2-12-1) but using sines instead of sigmoids, we achieved learning around 30 times faster (see table 4). Again, the assumption concerning the ranges of the weights appears to be correct.

3.6 Quadrants

It might be thought that not all functions are adequately approximated using sines as activation functions. Might there not be functions for which the sigmoids may be better suited than sines?.

To test this hypothesis, we designed a function that we call "Quadrants". In this problem the points of the first and third quadrants have target 1, and those of the second and fourth quadrants target 0. This problem seems to be tailor-made for sigmoids, as in order to obtain the correct classification it is only necessary to generate two hyperplanes. An activation function such as the sine, which can generate an infinite number of hyperplanes, does not seem suited to solving this type of problem.

The results are shown in table 5. Using the sine as activation function in the hidden layer and the sigmoid in the output layer, the training set was learnt with an architecture 2-2-1. For sigmoid functions, the training set was learnt with 4 hidden units (two more than in the previous case). Besides the advantage in terms of the number of hidden units, with the sine function learning is much faster than with the sigmoid function. Generalization is also

wrange	%lm	epochs
I-H:10 H-O:10	70	801.0
I-H:5 H-O:0.1	100	-
I-H:16 H-O:0.05	10	298.4
I-H:16 H-O:0.01	10	369.1
I-H:16 H-O:0.1	0	150.0

Table 4: Learning results for the "Separation problem" with different ranges of initial weights and learning rate 0.07. The percentage of local minima ('%lm') and the average number of learning epochs for succesful runs is shown.

architecture	wrange	%lm	epochs	generalization
2-2-1 sin-sig	I-H:1 H-O:1	9	433.4	100%
2-2-1 sin-sig	I-H:3 H-O:1	38	187.4	100%
2-2-1 sin-sig	I-H:5 H-O:1	23	192.9	100%
2-2-1 sin-sig	I-H:7 H-O:1	23	95.3	100%
2-2-1 sin-sig	I-H:7 H-O:0.1	23	144.8	100%
2-2-1 sin-sig	I-H:7 H-O:0.5	38	63.2	100%
2-4-1 sigmoid	I-H:1 H-O:1	17	826.5	97.5%

Table 5: Learning results for the "Quadrants" with different ranges of initial weights and learning rate 0.1 (sines-sigmoids) and 0.4 (sigmoids). The percentage of failures to learn the training set ('%lm'), the average number of learning epochs for succesful runs and the average percentage of correct generalization for succesful runs is shown. The initial range of bias is 1 in all cases.

better.

3.7 Diabetes

We performed another comparison with a problem of medical diagnosis, using the "Pima Indians Diabetes Database" produced in the Applied Physics Laboratory, Johns Hopkins University, 1988. The database comprises 768 women over the age of 21 resident in Phoenix, Arizona. In each example classification as diabetic is specified according to international criteria, plus eight variables considered relevant to the diagnosis, such as *plasma glucose concentration two hours after an oral tolerance test*, *body mass index* ($kg/height^2$), *age*, etc. All variables were numerical, and the database was complete.

To approach this problem we constructed a network with an 8-12-1 architecture. Three separate partitions of the set of examples were made, to construct three different training sets and their corresponding test sets. In each of the partitions the test sets contained 25% of the examples. The first partition was used to set the optimal learning rate and stop criterion. The stop criterion was fixed at successful learning of 80% of the training examples. Once these parameters were set, the network was trained and a test conducted with the sets of the other two partitions. Using sigmoid functions the generalization performance obtained was 72.3%, and with sine functions 78.8%. In further experiments to compare computational capacity, sigmoid functions were unable to learn more than 82% of the learning set correctly after 70,000 epochs, while the sine functions learnt more than 90% in around 1,500 epochs.

4 Other problems

In a learning task with chains generated by finite state automata (prediction of the following symbol) [Sopena & Alquézar 1994] and [Alquézar 1997] found learning to be ten times faster using sines than using sigmoids.

In a classification task with examples generated with a biconditional grammar, [Moliner & Tubau 1997] found that a network with sigmoid activation functions was unable to learn the training set. The same network with sine functions learnt the set extremely fast, and obtained a generalization of 99% with positive examples, and 98% with negative examples.

Finally, in a task involving the generation of fixed length sequences [Moliner & Sopena 1993] obtained a generalization of 98.2%, while the same architecture using sigmoid functions was unable to learn the task.

5 Conclusions

All the benchmarks confirmed the suppositions we presented in section 2 concerning the respective performance of the sine and sigmoid functions. A multilayer perceptron with sine activation functions learns much more quickly than one using sigmoid functions. Using standard BP, learning speed can be between 150 times faster (viz. "Negation N" and 500 times faster (viz. "Two spirals"). The speed is also greater than that obtained with the modified versions of BP proposed up to date.

The computational capacity of the networks with sine functions is also greater. All the functions that could not be learnt with sigmoid functions ("Two spirals","Separation problem", "Diabetes") were learnt easily with sine functions. Finally the capacity for generalization in all cases is greater, and in some cases considerably so.

The modification described here appears to be compatible with most modifications for BP suggested to date (for a review of the most important ones, see [Haykin 1994] and [Bishop 1995]). In future studies we aim to examine possible combinations with some of these modifications. It would also be interesting to analyze formally the properties of these types of network. Their similarities with Fourier series open up a range of interesting possibilities ([Lapedes & Farber 1987]).

References

[Alquézar 1997] Alquézar, R. (1997). Symbolic and Connectionist Learning Techniques for Grammatical Inferences. Ph.D. Tesis, Dept. LSI, UPC.

[Barron 1993] Barron, A. (1993). Universal Approximation Bounds for Superposition of a Sigmoidal Function. *IEEE Transactions on Information Theory, 39*, 930-945.

[Bishop 1995] Bishop, C.M. (1995). Neural Networks for Pattern Recognition. Oxford University Press Inc., New York.

[Bracewell 1986] Bracewell, R.N. (1986). The Fourier Transform and Its Applications. McGraw-Hill, Inc.

[Cybenko 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems, 2*, 303-314.

[Fahlman 1988] Fahlman, S.E. (1988). Faster Learning Variations: An empirical study. In Touretzky, D. et al. (ed.), *Proc. of the NN Model Summer School.* Morgan Kaufmann, San Mateo, Ca.

[Fahlman & Labiere 1990] Fahlman, S.E. and Labiere, C. (1990). The Cascade Correlation Learning Architecture. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann, San Mateo, Ca.

[Haykin 1994] Haykin, S. (1994). Neural Networks. A comprehensive foundation. Prentice Hall International, NJ.

[Hertz *et al.* 1991] Hertz, J., Krogh, A. and Palmer, R.G. (1991). Introduction to the Theory of Neural Computation. Reading, MA., Addison-Wesley.

[Jacobs 1988] Jacobs, R.A. (1988). Increased Rates of Convergence through Learning Rate Adaptation. *Neural Networks, 1*, 295-307.

[Lang & Witbrock 1988] Lang, K.J. and Witbrock, M.J. (1988). Learning to Tell Two Spirals Apart. *Proc. of the 1988 Connectionist Models Summer School*, Morgan Kaufmann.

[Lapedes & Farber 1987] Lapedes, A. and Farber, R. (1987). Nonlinear signal processing using neural networks: prediction and system modelling. LA-UR-87-2662 Los Alamos National Laboratory.

[Lippmann 1987] Lippmann, R.P. (1987). An Introduction to Computing with Neural Nets. *IEEE Acoustics, Speech, and Signal Processing*, 4-22.

[Lowe 1991] Lowe, D. (1991). What have Neural Networks to Offer Statistical Pattern Processing?. *Proc. of the SPIE Conference on Adaptative Signal Processing*, 460-471, San Diego, Ca.

[Moliner & Sopena 1993] Moliner, J.L. and Sopena, J.M. (1993). Variable binding using serial order in recurrent neural networks. In Mira, J. et al. (ed). *New Trends in Neural Computation.* LNCS, vol. 686. Springer-Verlag.

[Moliner & Tubau 1997] Moliner, J.L. and Tubau, E. (1997). Learning complex rules. *Proc. of the ASSC*, Pomona, Ca.

[Monte & Sopena 1993] Monte, E. and Sopena, J.M. (1993). El uso de la función de activación seno en las redes neuronales multicapa. Unpublished manuscript.

[Morita *et al.* 1990] Morita, M., Yoshizawa, S. and Nakano, K. (1990). Memory of correlated patterns by associative neural networks with improved dynamics. *Proceedings INNC'90, Paris, 2*, 868-871.

[Riedmiller & Braun 1993] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning. The RPROP algorithm. In Ruspini H. (ed.), *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, 586-591. San Francisco.

[Rumelhart *et al.* 1986] Rumelhart, D.E., Hinton, G.E. and Williams, R.J (1986). Parallel Distributed Processing, Vol 1. MIT Press.

[Sopena *et al.* 1993] Sopena, J.M., Monte, E. and Pérez, C. (1993). Un estudio comparativo de las funciones de activación periódicas y no periódicas en redes neuronales multicapa. Unpublished manuscript.

[Sopena & Alquézar 1994] Sopena, J.M. and Alquézar, R. (1994). Improvement of learning in recurrent networks by substituting the sigmoid activation function. *Proc. of the European Congress on ANN.* Sorrento, Italy.

[Tesauro & Janssens 1988] Tesauro, G. and Janssens, B. (1988). Scaling Relations in Backpropagation Learning. *Complex Systems, 2*, 39-44.

[Tollenare 1990] Tollenare, T. (1990). Supersab: Fast adaptive backpropagation with good scaling properties. *Neural Networks, 2* (5).

[Yang & Wanzhen 1993] Yang, L. and Wanzhen, Y. (1993). Backpropagation with Homotopy. *Neural Computation, 5*, 363-366.

[Wettschereck & Dietterich 1992] Wettschereck, D. and Dietterich, T. (1992). Improving the performance of radial basis function networks by learning center locations. *Advances in Neural Information Processing Systems 4*, 1133-1140. Morgan Kaufmann. San Mateo, CA.

[Yoshizawa *et al.* 1993] Yoshizawa, S., Morita, M., and Amari, S. (1993). Capacity of Associative Memory Using a Nonmonotonic Neuron Model. *Neural Networks, vol. 6*, 167-176.