
Pergamon

Neural Networks

Neural Networks 11 (1998) 259–270

Contributed article

Learning and approximation capabilities of adaptive spline activation function neural networks

Lorenzo Vecci, Francesco Piazza, Aurelio Uncini*

Dipartimento di Elettronica e Automatica—Università di Ancona, Via Brecce Bianche, 60131 Ancona, Italy

Received 8 October 1996; accepted 3 September 1997

Abstract

In this paper, we study the theoretical properties of a new kind of artificial neural network, which is able to adapt its activation functions by varying the control points of a Catmull–Rom cubic spline. Most of all, we are interested in generalization capability, and we can show that our architecture presents several advantages. First of all, it can be seen as a sub-optimal realization of the additive spline based model obtained by the reguralization theory. Besides, simulations confirm that the special learning mechanism allows to use in a very effective way the network's free parameters, keeping their total number at lower values than in networks with sigmoidal activation functions. Other notable properties are a shorter training time and a reduced hardware complexity, due to the surplus in the number of neurons. © 1998 Elsevier Science Ltd. All rights reserved.

Keywords: Spline neural networks; Multilayer perceptron; Generalized sigmoidal functions; Adaptive activation functions; Function shape autotuning; Generalization; Neural networks

1. Introduction

Recently in the neural network community, a new interest in adaptive activation functions has arisen. In fact, such a strategy seems to provide better fitting properties with respect to classical architectures with sigmoidal neurons. Our purpose here is to study a new kind of adaptive activation function from a theoretical point of view and to justify the observed behavior on some significant test problems, stressing those properties related to some of the most important theories of learning.

The simplest solution we can imagine consists in involving gain a and slope b of the sigmoid $a(1 - e^{-bx})/(1 + e^{-bx})$ in the learning process; in Chen and Chang (1996) a comparison with classical multi-layer perception (MLP) to model static and dynamical systems is reported, showing that an adaptive sigmoid in a single hidden layer structure leads to an improved data modeling. In Reed et al. (1995) it is pointed out that a proper sigmoid scaling is related to training with jitters, a well-known trick to achieve a good generalization capability. A different approach is based on the use of polynomial functions (see Piazza et al., 1992), which allow us to reduce the size of the network and, in particular, connection complexity; then, the digital implementation of the activation function through a LUT (look-up-table) keeps the overall complexity under control. Drawbacks with this solution arise with the non-boundedness of the function (non-squashing), and with the adaptation of the coefficients in the learning phase. In Piazza et al. (1993) the direct adaptation of the LUT coefficients is tried: this time the problems are a difficult learning process due to the large number of free parameters and the lack of smoothness of the neuron's output. These are also the main reasons for the introduction of Hermite polynomials as substitutes for the so called supersmoother in the Projection Pursuit Learning (PPL) approach of Hwang et al. (1994); we are not specifically interested in a constructive learning algorithm [we retain classical backpropagation (BP)], but we notice a good agreement with PPL because of the relevance given to adaptive activation functions as sources of parsimony in terms of hidden units.

The solution discussed in this paper makes use of spline based activation functions whose shape can be modified through some control points. In particular we'll show how our architecture is related to cubic splines as obtained by the application of regularization theory. In fact, our main goal is to demonstrate that an intelligent use of the activation function can reduce hardware complexity (see Campolucci et al.,

* Request for reprints should be sent to Aurelio Uncini. Phone: +39 (71) 220 4841; Fax: +39 (71) 220 4464; e-mail: aurel@eealab.unian.it.

PII S0893-6080(97)00118-4

1996; Piazza et al., 1996; and Guarnieri et al., 1995), while, at the same time, improving generalization ability (which is the main theoretical aspect we are interested in now). Given a training set $T_N = \{(\mathbf{x}_1,t_1),...,(\mathbf{x}_N,t_N)\}$ (let's consider for sake of simplicity $\mathbf{x}_i \in X \subseteq R_n$ and $t_i \in Y \subseteq R$), there are infinite possible hypersurfaces able to give a good approximation of its elements, but not all of them can be judged in the same way. Of course, we want the approximator to track the target function also on points not belonging to T_N: that is what we mean by generalization ability, a property of central importance studied by all the most important theories about learning.

Regularization theory (see Poggio and Girosi, 1990a, b) offers a synthesis-oriented approach, as it allows us to build networks which are the exact solution of a minimization problem involving a compromise between the error on the samples in T_N and the smoothness of the approximation: for smoothness we mean that the neural network should be able to avoid non-essential elements in the data, as high frequency components in the domain X due to noise, and so it shouldn't give origin to useless oscillations which give a non-smooth aspect to the hypersurface. The choice of the regularization parameter (introduced in Section 2.1) also governs the compromise between bias and variance, (see Geman et al., 1992 and Wolpert, 1995): a small λ is responsible for a close fitting of the training and of a relatively large variance of the final approximation while a large λ is the cause of a large bias. An equivalent problem is encountered when choosing the number of free parameters in many non-parametric regression techniques.

Regularization theory is the basis of many techniques whose goal is a better generalization: some of them are addressed to the training set (i.e., training with jitters, see Reed et al., 1995); on the other hand, we are interested in deriving an architecture which embodies some regularity characteristics in its own activation function much better than the classical sigmoid can do. So, after seeing cubic splines in regularization theory and discussing some drawbacks in their use (Section 2.1), we will explain our strategy (Section 2.2); then we will show how we can control the smoothness of the surface designed by the network (Section 2.3).

In Section 3, we will expose the results of some simulations on simple two-dimensional functions, on a more complex nonlinear system and on a pattern recognition problem: we will verify the good generalization ability expected according to the theory and we will notice some other advantages of this kind of network. In particular we will see that the possibility of tuning the activation function determines the reduction of the number of hidden units together with the number of free parameters, which is very important from a theoretical point of view, as pointed out in Guyon et al. (1992) and in Moody and Utans (1994). Statistical learning theories, like probably approximated correct (PAC) or Bayesian, all try to evaluate the generalization error: although there are not many results addressing particular architectures (see Holden and Rayner, 1995 and Niyogi and Girosi, 1996), for binary classification problems in Amari (1993) and Amari and Murata (1993), theorems have been demonstrated which give an asymptotic upper bound of N_{fp}/N, where N_{fp} is the total number of free parameters. This is not a very strict bound (in general the error is lower), and anyway it gives a warning about the choice of a too large number of adaptive coefficients, which could cause overfitting. It may seem that the introduction of control points in the activation function leads to a larger value for N_{fp}, but this is not true. In fact, in practical situations, we need a few control points for each neuron to obtain a good approximation and, at the same time, we greatly reduce the number of hidden units, so eliminating a lot of connection weights (especially when the input space has a high dimensionality): as a consequence, in single hidden layer structures, our strategy allows the reduction of the overall number of free parameters.

2. Cubic splines as activation function

2.1. Cubic splines in regularization theory

The reconstruction of a hypersurface from the knowledge of a finite set, T_N, of samples is a typical ill-posed problem; in general, there are many possible functions with good approximation capabilities on the pairs $(\mathbf{x}_i,t_i)$ and not all of them can be considered in the same way. In fact, the presence of noise in the measurements induces, in the space X, high-frequency components that have to be filtered by the neural model, because they are not useful for the identification of the underlying target but are only a disturbance. At the same time, in those regions of X in which we don't have many samples, the neural network should give a smooth approximation of the few available data, avoiding unjustified oscillations.

Regularization theory offers a way to choose a compromise between data fitting and smoothness, through a regularizing term added to the classical squared error and weighted by a constant:

$$H(f) = \sum_{i=1}^{N} [t_i - f(\mathbf{x}_t)]^2 + \lambda ||Pf||^2 \tag{1}$$

where $H(f)$ represents the functional to be minimized. The stabilizer P is the differential operator determining the kind of smoothness and the shape of the approximator, while $\|.\|$ is a suitable norm. It is well known that the minimization of Eq. (1) leads to

$$f(\mathbf{x}) = \sum_{i=1}^{N} c_i G(\mathbf{x} - \mathbf{x}_i); \tag{2}$$

G is the Green's function corresponding to the operator P and c_i are coefficients determined by solving the $N \times N$ linear system $(\mathbf{G} + \lambda\mathbf{I})\mathbf{c} = \mathbf{t}$, where $\mathbf{G}$ is the Green's matrix,

c is the column vector of the coefficients and **t** is the column vector of the targets t_i.

In particular, we are interested in the one-dimensional stabilizer

$$||Pf||^2 = \int_R \left[\frac{d^2 f(x)}{dx^2}\right]^2 dx \tag{3}$$

which corresponds to the kernel $G(x) = |x|^3$. For the multi-dimensional case, in Girosi et al. (1995) it is shown that we can use the same stabilizer, just decomposing the function f in the sum of n functions, each in charge of one component of the vector **x**

$$f(\mathbf{x}) = \sum_{j=1}^{n} f_j(x_j). \tag{4}$$

Then, the overall kernel is

$$\bar{G}(x) = \sum_{j=1}^{n} \mu_j G(x_j) = \sum_{j=1}^{n} \mu_j |x_j|^3, \tag{5}$$

where $_j$, $j = 1,...,n$, are constants. The final aspect of the approximating function is

$$f(\mathbf{x}) = \sum_{i=1}^{N} c_i \sum_{j=1}^{n} \mu_j G(x_j - x_{ij}); \tag{6}$$

the symbol x_{ij} indicates the jth component of the ith input $\mathbf{x}_i$ in T_N. An important extension of the previous function involves a change in the system of coordinates for the space X; as reported in Girosi et al. (1995), the choice of a proper ''point of view'' can be important when representing a multivariate function as the sum of a number of functions equal to the dimension of the input space. Calling $\mathbf{w}_j$, $j = 1,...,n$, the vectors which determine the axis of the new system and α_{ij} the new centers in such a system, we can write, inverting the order of summation of Eq. (6),

$$f(\mathbf{x}) = \sum_{j=1}^{n} \mu_j \sum_{i=1}^{N} c_i G(\mathbf{w}_j \mathbf{x} - \alpha_{ij}); \tag{7}$$

that is the starting point of our considerations.

First of all, we need to evaluate the fixed parameters μ_j; with no a priori assumption we may use data-driven procedures, i.e., cross-validation, which can be computationally expensive when applied to large nonlinear models like most neural networks. Then a $N \times N$ linear system must be solved in order to find the coefficients c_i, $i = 1,...,N$; in real applications, the number N can be large and the problems related to the inversion of the matrix $(\mathbf{G} + \lambda\mathbf{I})$ can become quite hard. The large value of N is an obstacle for hardware implementation, too, because we should use nN kernels (N kernels for each of the n directions), which means nN neurons, a lot of connections and, in the case of VLSI implementation, a large silicon area. Of course, it is possible to reduce the number of centers from N to N' ($N' < N$) using a sub-optimal solution of the minimization of Eq. (1), but here we will take a different approach.

Our idea consists in realizing a neuron with a more complex activation function than the sigmoid, able to reproduce the shape of a whole cubic spline along the directions specified by $\mathbf{w}_j$, $j = 1,...,n$.

$$\varphi(\mathbf{w}_j \mathbf{x}) = \sum_{i=1}^{N} c_i |\mathbf{w}_j \mathbf{x} - \alpha_{ij}|^3 \quad j = 1, ..., n \tag{8}$$

Then $f(\mathbf{x})$ can be written as

$$f(\mathbf{x}) = \sum_{j=1}^{n} \mu_j \varphi_j(\mathbf{w}_j \mathbf{x}), \tag{9}$$

and the final architecture is shown in Fig. 1. Now μ_j, and the components of $\mathbf{w}_j$, for all the indexes j, can be found by backpropagation, thus solving the problem of the optimal set of the parameters μ_j and of the ideal system of coordinates (although we can get trapped in local minima). The open question is about φ_j. Once again, the exact implementation of Eq. (8) would require the knowledge of all the coefficients c_i, so we choose a different solution, that is using a cubic spline of simpler structure. Its main characteristics are the adaptation of its shape through some control points and a suitable degree of smoothness. Notice that in our implementation a bias parameter w_{j0} similar to the one used in sigmoidal neurons has been introduced and so we will deal with a function $\varphi_j(\mathbf{w}_j \mathbf{x} + w_{j0})$. The last point to discuss is the approximation capability of the function in Eq. (8): of course, it will behave well on targets which are likely to have an additive structure, but, in general, a number of hidden units equal to the dimension of the input space is not enough to obtain universal approximation of continuous functions on a compact set. However, we can extend the idea of a neuron with a cubic and smooth activation function to architectures involving a larger number of hidden units (or even more than one hidden layer), though they cannot be directly derived from regularization theory.

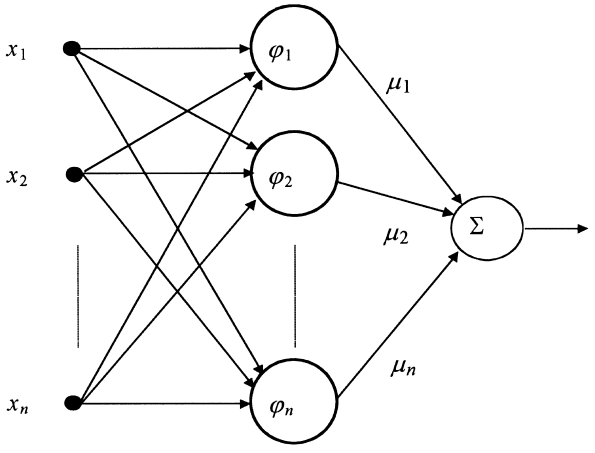

Fig. 1. Architecture deriving from Eq. (9): the functions φ_j, $j = 1,...,n$, are the cubic splines of Eq. (8).

2.2. The cubic spline based adaptive activation function

Referring to the papers by Campolucci et al. (1996) and Guarnieri et al. (1995) for further details, here we give some notes on such realization of an adaptive activation function. As we have anticipated in Section 2.1, the goal is to give a global approximation of the curve drawn by the functions φ_j, $j = 1,...,n$, using a structure as tractable as possible. In Eq. (8) we find a spline with $N + 1$ tracts: each of them is described by a different combination of the coefficients c_i, because of the change in the sign of the kernels at α_{ij}. We have chosen to represent the activation functions through the concatenation of even more local spline basis functions, controlled by only four coefficients. As we want to keep the cubic characteristic, we have used a Catmull–Rom cubic spline. Using this type of spline we could exactly reproduce function Eq. (8), but, of course, this is not the cheapest solution, as reported in Appendix B, so we will take a different approach. Referring to Fig. 2, the ith tract is expressed as

$$\mathbf{F}_i(u) = \begin{bmatrix} F_{x,i}(u) \\ F_{y,i}(u) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{Q}_i \\ \mathbf{Q}_{i+1} \\ \mathbf{Q}_{i+2} \\ \mathbf{Q}_{i+3} \end{bmatrix} \tag{10}$$

where $u[0,1]$ and $\mathbf{Q} = (q_x, q_y)$. Such a spline interpolates the points $\mathbf{Q}_{i+1}(u = 0)$ and $\mathbf{Q}_{i+2}(u = 1)$ and has a continuous first derivative, which is useful for the backpropagation-like learning algorithm (see Appendix A). The second derivative is not continuous only at the knots. In general, Eq. (10) represents a curve: to obtain a function we have ordered the x-coordinates according to the rule $q_{x,i} < q_{x,i+1} < q_{x,i+2} < q_{x,i+3}$.

To find the value of the local parameter u, we have to solve the equation $F_{x,i}(u) = x_0$, where x_0 is the activation of the neuron: this is a third degree equation, whose solution can make the numerical burden of the learning algorithm heavier. The easiest alternative consists in setting the control points uniformly spaced along the x-axis (Δx is the step): this choice allows to reduce the third degree polynomial $F_{x,i}(u)$ to a first degree polynomial and the equation for u becomes linear.

$$F_{x,i}(u) = u\Delta x + q_{x,i+1} \tag{11}$$

Now we can calculate the output of the neuron by $F_{y,i}(u)$. There is another reason not to make the x-coordinates of the control points adaptive: in fact, as we have pointed out in the introduction, a too-large number of free parameters is the main cause for the overfitting of the training samples: so, if we use many tracts in building the activation function and let them move freely, the neural model will fit also the noise. Then the fixed parameter Δx is the key tool for smoothness control.

As we have decided to adapt only the y-coordinates of the spline knots, we must initialize them before starting the backpropagation-style learning: to this aim, we take, along the x-axis, $P + 1$ uniformly spaced samples from a sigmoid or from another function assuring universal approximation capability (see Cybenko, 1989 and Stinchcombe and White, 1989): that is why we use sometimes the acronym GS, standing for ''Generalized Sigmoid''. Outside the sampling interval the neuron's output will be held constant at the values $q_{y,1}$, for the negative x-coordinate, and $q_{y,P-1}$ for the positive x. In the following, for simplicity's sake, we will indicate the y-coordinates of the control points without the index y. For the whole network the acronym ASNN, ''Adaptive Spline Neural Networks'', has been chosen.

2.3. Smoothness control of the activation function

The Catmull–Rom spline based activation function can be seen as a sub-optimal and adaptive realization f_j of the cubic spline φ_j obtained through the kernels $|x|^3$. The problem is now the control of the degree of smoothness. The most suitable measure of this property is given exactly by Eq. (3), where the integration variable x is replaced by the activation of the neuron. Considering only the single hidden

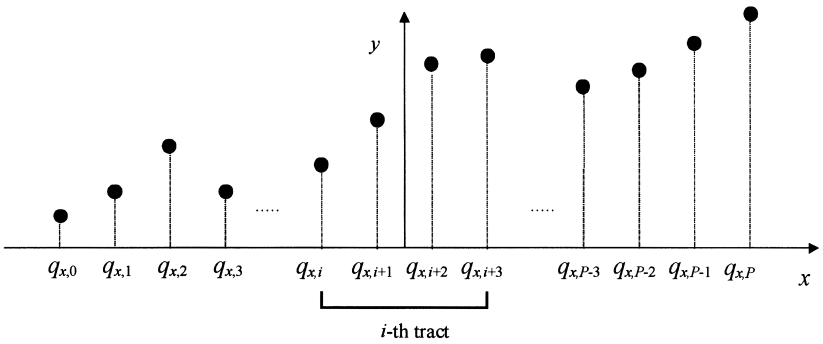

Fig. 2. Control points of the Catmull–Rom spline based activation function: along the x-axis there is a fixed step Δx. The ith tract starts with $q_{x,i}$ and ends with $q_{x,i+3}$, but the controlled interval is only between $q_{x,i+1}$ and $q_{x,i+2}$.

layer structure with Σ a neuron in the output, it holds that

$$s_j = \sum_{k=1}^{n} w_{jk}x_k + w_{j0}; \tag{12}$$

with

$$z_j = \frac{s_j}{\Delta x} + \frac{P-2}{2} \tag{13}$$

$$a_j = \lfloor z_j \rfloor$$

$$u_j = \frac{s_j - q_{jx,(a_j+1)}}{\Delta x} = z_j - a_j$$

where the symbol $\lfloor . \rfloor$ is the floor operator. We find for each neuron j the local tract a_j which s_j belongs to, and the local coordinate u_j. Then we can write, assuming a generic number H of hidden units,

$$f(\mathbf{x}) = \sum_{j=1}^{H} \mu_j f_j \left(\sum_{k=1}^{n} w_{jk}x_k + w_{j0} \right) \tag{14}$$

$$f_j \left(\sum_{k=1}^{n} w_{jk}x_k + w_{j0} \right)$$

$$= \begin{cases} q_{j,1} & \text{if } s_j \le q_{jx,1} \\ \frac{1}{2}\left(-q_{j,(a_j)} + 3q_{j,(a_j+1)} - 3q_{j,(a_j+2)} + q_{j,(a_j+3)} \right) \left(\frac{s_j - q_{jx,(a_j+1)}}{\Delta x} \right)^3 \\ \quad + \frac{1}{2}\left(2q_{j,(a_j)} - 5q_{j,(a_j+1)} + 4q_{j,(a_j+2)} - q_{j,(a_j+3)} \right) \left(\frac{s_j - q_{jx,(a_j+1)}}{\Delta x} \right)^2 + \frac{1}{2}\left(-q_{j,(a_j)} + q_{j,(a_j+2)} \right) \left(\frac{s_j - q_{jx,(a_j+1)}}{\Delta x} \right) + q_{j,(a_j+1)} \\ \quad \text{if } q_{jx,1} < s_j < q_{jx,P-1} \\ q_{jx,P-1} & \text{if } s_j \ge q_{jx,P-1} \end{cases}$$

Then, let us consider

$$\int_R \left[\frac{\mathrm{d}^2 f_j(s_j)}{\mathrm{d}s_j^2} \right] \mathrm{d}s_j = \frac{1}{\Delta x^3} \sum_{p=0}^{P-3} \big[3\left(-q_{j,p} + 3q_{j,p+1} - 3q_{j,p+2} + q_{j,p+3} \right)^2 + 3\left(-q_{j,p} + 3q_{j,p+1} - 3q_{j,p+2} + q_{j,p+3} \right) \cdot \left(2q_{j,p} - 5q_{j,p+1} + 4q_{j,p+2} - q_{j,p+3} \right) + \left(2q_{j,p} - 5q_{j,p+1} + 4q_{j,p+2} - q_{j,p+3} \right)^2 \big]$$

$$= \frac{1}{\Delta x^3} \sum_{p=0}^{p-3} \left(\Delta \mathrm{I}_{j,p}^2 + 3\Delta \mathrm{II}_{j,p}^2 + \Delta \mathrm{III}_{j,p}^2 - 3\Delta \mathrm{I}_{j,p}\Delta \mathrm{II}_{j,p} + \Delta \mathrm{I}_{j,p}\Delta \mathrm{III}_{j,p} - 3\Delta \mathrm{II}_{j,p}\Delta \mathrm{III}_{j,p} \right) \tag{15}$$

with $\Delta \mathrm{I}_{j,p} = q_{j,p+1} - q_{j,p}$, $\Delta \mathrm{II}_{j,p} = q_{j,p+2} - q_{j,p+1}$ and $\Delta \mathrm{III}_{j,p} = q_{j,p+3} - q_{j,p+2}$. Now, it can be noticed that the smoothness of the activation function depends on the distance between the y-coordinates of the control points and on the fixed step Δx. The values of $q_{j,p}, j = 1,...,H, p = 2,...,P-2$, are adapted during the learning process and so the only way to keep them close is to add to the error function an extra term penalizing a large distance between the y-coordinates on the same tract. We experimented this strategy in different situations, but only in the case of a particular one-dimensional function with very sparse training data improved results were obtained: in general, the experiments showed that this kind of constraint may represent a too severe limitation. On the other hand, it is much easier to tune the smoothness using Δx: with a suitable step we can find a good degree of regularity of the function realized by the network. Some tests we carried on indicate that too small Δx (< 0.5) can easily result in an overfitted approximation, but above a threshold, also if there is an optimal value, the generalization error is rather insensitive to Δx: this means that the distance between control points can be chosen without complex procedures like cross-validation, but just by applying a manual tuning. In Table 1, where the generalization ($\langle g.e. \rangle$) errors are reported in dB ($10\mathrm{Log}\,(\Sigma(t_i - \mathrm{f}(\mathbf{x}_i))^2/N)$), we see an example relative to the problem of function approximation varying the Δx (see Hassoun, 1996)

$$g(x) = \frac{(x-2)(2x+1)}{1+x^2}. \tag{16}$$

The networks used in this test have one input two hidden spline neurons and one linear output. The training consists in 50 samples in the interval $[-5,5]$ and large fixed impulsive noise (outliers) is superimposed to the desired outputs. The noise has uniform distribution in the range $[-5,5]$ and added with probability 0.3 (only about the 30% of the training samples are corrupted): the training lasts 1000 epochs. In Fig. 3a–c, the dash–dotted line represents the target function. The solid line in Fig. 3a represents the noisy training samples. In Fig. 3b the approximation reached using neurons having $\Delta x = (0.4,0.6,0.8)$ is reported, while Fig. 3c shows the results of the approximation using neurons having $\Delta x = (1,2,4)$. It is evident that when the Δx value is greater than a threshold (0.5 in this case) the results are quite equivalent. This property, also tested in the more complex

Table 1
Simulation results for fitting function Eq. (16). The learning is stopped after 1000 iterations using $\mu_w = 0.005$, $\mu_q = 0.005$. For each Δx value five networks are trained

Net	Δx	$\langle g.e. \rangle$ (SD)
s_04	0.4	− 8.43 (2.06)
s_06	0.6	− 9.46 (0.06)
s_08	0.8	− 9.32 (0.02)
s_1	1	− 9.28 (0.02)
s_2	2	− 9.16 (0.08)
s_4	4	− 9.08 (0.05)

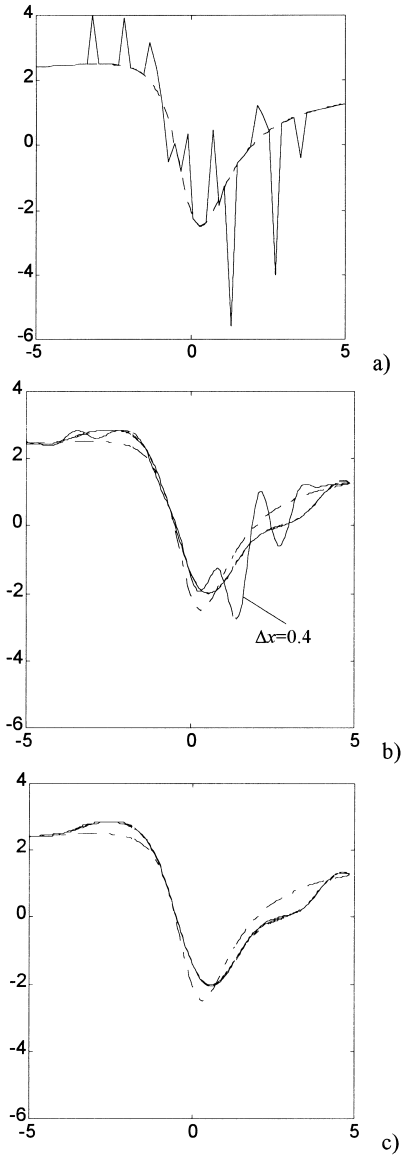

Fig. 3. Approximations of function Eq. (16), the dash–dotted lines represents the target function. (a) The solid line represents the 50 training points corrupted by large additive impulsive noise (outliers). (b) The solid lines represent the 200 points networks' output with $\Delta x = (0.4,0.6,0.8)$. (c) The solid lines represent the 200 points networks' output with $\Delta x = (1,2,4)$.

situation of Section 3.3, is very useful in real applications where validation techniques such as cross-validation are not practicable, e.g., too-small training sets. Let N_m be the total number of parameters whose values have been modified during the overall learning process (this value can be calculated by an off-line analysis after the learning procedure); in general, it is not possible to determine an analytical relationship between N_m, the initial values of the connection weights, and the values of the training inputs, although an unknown relation like $N_m \propto \psi(\mathbf{W},\mathbf{X},\Delta x)$, with $\mathbf{W}$ representing the weights matrix of the input layer and $\mathbf{X}$ the net's inputs, may be hypothesized. When experimenting with small initial weights, a behavior like the one of Fig. 4 can be noticed, that is, after a few learning iterations, the span of

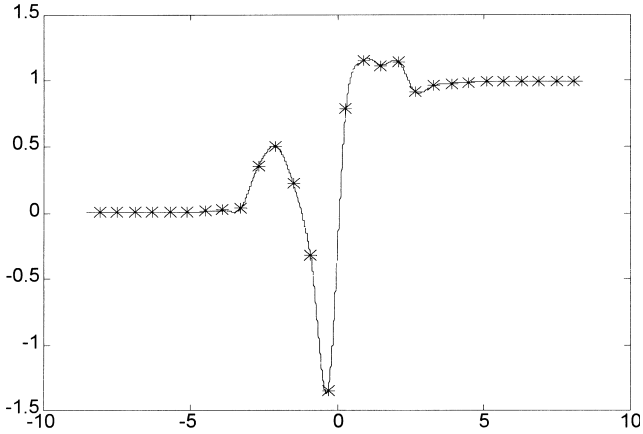

Fig. 4. Activation function of the spline based neuron approximating function Eq. (16): of the 28 control points (the stars), only 10 have been adapted by the learning algorithm ($\Delta x = 0.4$).

the spline used by each neuron converges to a small number of control points (if Δx is large enough) centered on the origin.An interesting property due to the learning algorithm based on a modified backpropagation can be observed: we can easily verify that $N_m \leq N_{fp}$ which, in some sense, can be viewed as a self-regularization according to Moody's theory. On the other hand, algorithms which simultaneously perturb all the free parameters ($N_m = N_{fp}$), are not so convenient for the architecture we present neither from a computational point of view nor for what concerns regularization.

3. Simulations and discussion

3.1. Test functions

In this section we present the results of some simulations on simple test functions. We aim at comparing the classical sigmoid with the spline based activation function on architectures having a single hidden layer and a Σ-neuron in the output. We will focus on two main aspects, that are generalization capability and the control of the number of free parameters. First of all, it is worth noting that, while in a sigmoidal network all the adaptive parameters are updated at every step of the learning algorithm, the situation in our networks is quite different. Referring to Appendix A for further details, it can be observed that, at each learning step, all the connection weights change their value, but only four control points in each neuron are updated. This strategy can be considered an application of the principle of minimal disturbance (see Widrow and Lehr, 1990), stating that the parameters of an adaptive system should be disturbed as little as possible when a new input data is presented. As every input to the network, after the weighted summation, involves only a limited segment of the x-axis of the activation function, we should move only the control points relative to that segment, without altering the shape of the neuron's output in other intervals.

Table 2
Simulation results for fitting function Eq. (17)

Net	N_l	N_m	−20 dB			−22 dB			−24 dB			50 000 epochs	
			⟨*l.e.*⟩	⟨*g.e.*⟩	$\sigma^2_{g.e.}$	⟨*l.e.*⟩	⟨*g.e.*⟩	$\sigma^2_{g.e.}$	⟨*l.e.*⟩	⟨*g.e.*⟩	$\sigma^2_{g.e.}$	⟨*g.e.*⟩	$\sigma^2_{g.e.}$
N_4	17	17	–	–	–	–	–	–	–	–	–	–	–
N_6	25	25	–	–	–	–	–	–	–	–	–	–	–
N_8	33	33	16 889	−10.90	$1.58\cdot10^{-3}$	26 635	−11.51	$1.39\cdot10^{-3}$	–	–	–	−11.64	$1.26\cdot10^{-3}$
N_10	41	41	10 868	−10.60	$2.98\cdot10^{-3}$	14 093	−10.76	$4.32\cdot10^{-3}$	25 757	−10.52	$5.31\cdot10^{-3}$	–	–
N_15	61	61	5551	−10.88	$2.31\cdot10^{-3}$	8723	−11.82	$1.31\cdot10^{-3}$	19 059	−12.26	$9.98\cdot10^{-4}$	–	–
s_2	17	23	9403	−15.12	$2.65\cdot10^{-7}$	–	–	–	–	–	–	−15.68	$3.46\cdot10^{-8}$
s_4	33	43	3280	−14.79	$9.07\cdot10^{-5}$	11 067	−15.98	$1.79\cdot10^{-5}$	–	–	–	−16.22	$1.17\cdot10^{-5}$

A second observation is that an arbitrary number of control points can be initially taken (the value 28 in our tests was determined empirically), because just some of them (in general in an interval centered on the origin) are actually moved during the learning, while the others are never involved by any input data and simply join the activation function to its asymptotes: thus, the method is not deeply influnced by this choice except that if too few control points are taken more hidden neurons may be required.

The special behavior of the adaptation mechanism is shown for the one-dimensional example of function Eq. (16): if we have a look at Fig. 4, where the activation function of the adaptive hidden neuron is represented, we can see that many of the control points were not updated by the learning algorithm because no input data involved their intervals. So the number of coefficients which were changed by backpropagation is only 14. This example teaches we must not be misled by the fact that we took originally 28 points to control the output of a neuron: the learning mechanism is able to select those ones which are effectively useful to the approximation; in this sense the architecture seems to be able to fit the effective number of free parameters (see Moody, 1992) to the training set it has to deal with. This is a very welcome property: as we pointed out in the introduction the matching between the size of network and the complexity of the problem is the key for a good generalization performance.

The results of further simulations on two-dimensional functions are arranged in Tables 2 and 3. The first column includes the different types of networks which were tested: *N_x* stands for a classical MLP with *x* hidden units, while s_*x* is an ASNN with *x* hidden neurons. The first column contains the N_l, that is the number of parameters updated at every learning cycle, while the second is relative to N_m. As we have pointed out at the beginning of the paragraph, for the sigmoidal MLP we have $N_l = N_m$, while for GS neurons the situation is different: in fact N_l is simply obtained by summing four control points for each GS neuron to the number of connection weights, while N_m can be calculated only at the end of the training, this time adding to the number of connection weights the control points which have been effectively moved for every GS neuron. The tests are performed stopping the training when −20 dB, −22 dB and −24 dB in the learning phase are reached (⟨*l.e.*⟩ is the number of necessary learning epochs): T_{50} is made of 50 samples lying in the square $[-1,+1][-1,+1]$, obtained by a uniform probability density with zero mean and 0.1 variance Gaussian noise added to the desired outputs. The criterium for the computation of generalization error mean, expressed in dB, and variance (⟨*g.e.*⟩ and $\sigma^2_{g.e.}$) is based on the inizialization of 20 networks: if at least half of this group reaches the threshold for training stop (−20 dB or −22 dB or −24 dB), ⟨*g.e.*⟩ and $\sigma^2_{g.e.}$ are calculated averaging on the best 10 networks over an equally spaced grid of 441 points in $[-1,+1][-1,+1]$, otherwise no values are reported in the corresponding columns. If 50% of the networks fail to reach −20 dB, the architecture is considered unsuccessful; otherwise training is carried on and the remaining two thresholds are tested. It is also considered the situation after 50 000 epochs, except for networks that can even reach −24 dB.

The first function, plotted in Fig. 5a, is

$$g(x,y) = \sin(2\pi x) + 4(y-0.5)^2 \tag{17}$$

while the second test is on the two-dimensional Gabor function

$$g(x,y) = \mathrm{e}^{-(x^2+y^2)}\cos[0.75\pi(x+y)] \tag{18}$$

The spline networks are chosen with $\Delta x = 0.5$ and

Table 3
Simulation results for fitting function Eq. (18)

Net	N_l	N_m	−20 dB			−22 dB			−24 dB			50 000 epochs	
			⟨*l.e.*⟩	⟨*g.e.*⟩	$\sigma^2_{g.e.}$	⟨*l.e.*⟩	⟨*g.e.*⟩	$\sigma^2_{g.e.}$	⟨*l.e.*⟩	⟨*g.e.*⟩	$\sigma^2_{g.e.}$	⟨*g.e.*⟩	$\sigma^2_{g.e.}$
N_8	33	33	8458	−14.89	$2.37\cdot10^{-4}$	–	–	–	–	–	–	−17.30	$1.15\cdot10^{-4}$
N_11	45	45	3186	−14.02	$2.60\cdot10^{-4}$	29 936	−17.34	$5.45\cdot10^{-5}$	–	–	–	−17.80	$6.09\cdot10^{-5}$
N_15	61	61	2406	−13.31	$4.15\cdot10^{-4}$	20 740	−17.53	$4.60\cdot10^{-5}$	–	–	–	−16.90	$3.47\cdot10^{-4}$
s_4	33	43	3194	−17.85	$7.94\cdot10^{-6}$	13 476	−19.68	$2.09\cdot10^{-5}$	–	–	–	−13.94	$1.26\cdot10^{-4}$

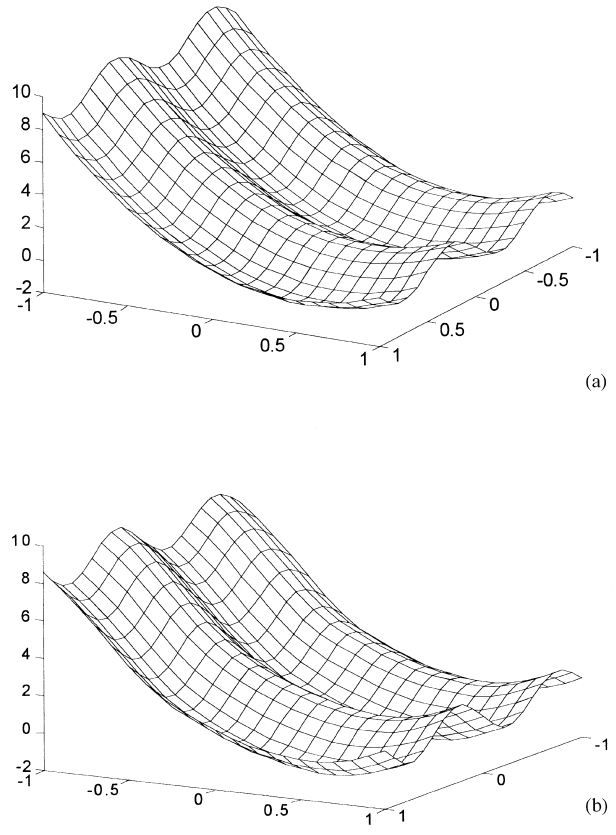

Fig. 5. (a) The additive function of Eq. (17). (b) Approximation of function Eq. (17) using two spline based hidden neurons: it can be noticed a good agreement with the target.

compared to sigmoidal networks on the basis of N_l and N_m; a classical MLP architecture with more free parameters ($N_l = N_m = 61$) is also presented. Also if the input space is two-dimensional, the simulations on function Eq. (17) are an important example of how some extra dimension can be useful to improve the quality of the approximation, also if the target has a simple additive structure; for function Eq. (18) a nonlinear transformation over two dimensions is not sufficient and four hidden neurons are considered to obtain a good performance on the data.

The spline network with two hidden neurons should be compared to sigmoidal networks with four and six hidden neurons which are not able to reach -20 dB and so are discarded. It is clear from $\langle g.e. \rangle$ columns that our architecture has a lower generalization error and that it is also more convenient for what concerns implementation problems: in fact the size in terms of hidden units is greatly reduced.

A very remarkable feature of GS neurons is that they seem to produce a more tractable error surface, which leads to a narrower variance of the generalization error after the learning. Although the error function depends on many variables and it is quite difficult to study in theory, the experimental results give the impression of a smooth shape, on which it is easier to end the training close to the same point of minimum. This fact is particularly evident when the training is stopped at an early stage (when -20 dB are reached) and in the case of the additive target function; only for the Gabor function, after 50 000 epochs values of $\sigma^2_{g.e.}$ for sigmoidal networks have generally a better behavior, but it is also clear that the spline network is overtrained.

A related advantage in the use of GS neurons is the training speed: N_l and N_m being equal, our architecture is often faster in reaching the thresholds and it is outpaced only by the network with 15 hidden sigmoids which has many more adaptive parameters. Adaptive activation functions reveal themselves to be more interesting in applications where learning speed is a key factor, together with an economical architecture in terms of hidden units.

3.2. Back and Tsoi system

Now we are going to test the proposed architecture on a more complex problem, that is the identification of a system

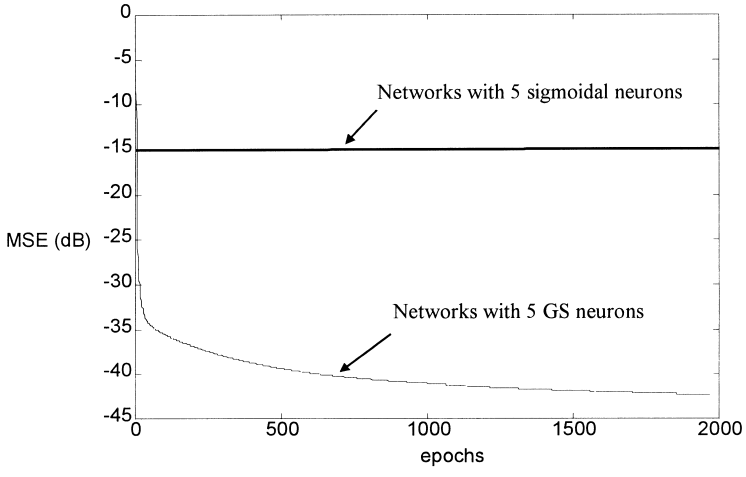

Fig. 6. Training error for the Back and Tsoi system: comparison of architectures with the same number of hidden units.

proposed by Back and Tsoi (1993), made of a IIR linear filter followed by an instantaneous nonlinearity.

$$z[t] = 0.0154u[t] + 0.0462u[t-1] + 0.0462u[t-2] + 0.0152u[t-3] + 1.99z[t-1] - 1.572z[t-2] + 0.4583z[t-3] \quad (19)$$

$$y[t] = \sin\left(\frac{\pi}{2}z[t]\right)$$

The temporal dynamic is obtained externally feeding into the network $u[t]$,$u[t-1]$,...,$u[t-5]$ and $y[t-1]$,...,$y[t-5]$. Fixing arbitrary initial conditions for u and y, we track the behavior of the system and take 1000 samples to be used as the training set; a second group of independently obtained 1000 samples is the test set. Experiments with just five hidden neurons give excellent results both in terms of training error and of generalization error (Figs. 6 and 7 and Table 4) with a total number of 96 coefficients updated by backpropagation. The line at −15 dB in Fig. 6 represents the limit of the best performance reached by networks with five sigmoidal hidden units. If we want to obtain a MSE of approximately −40 dB in no more then 15 000 epochs, we are forced to use 20 sigmoidal hidden neurons, more or less: this means an architecture involving 261 free parameters all of which must be adapted leading to a very long training time and to a more dangerous situation as far as generalization is concerned; besides the architecture is quite expensive if compared to the solution using five spline based neurons.

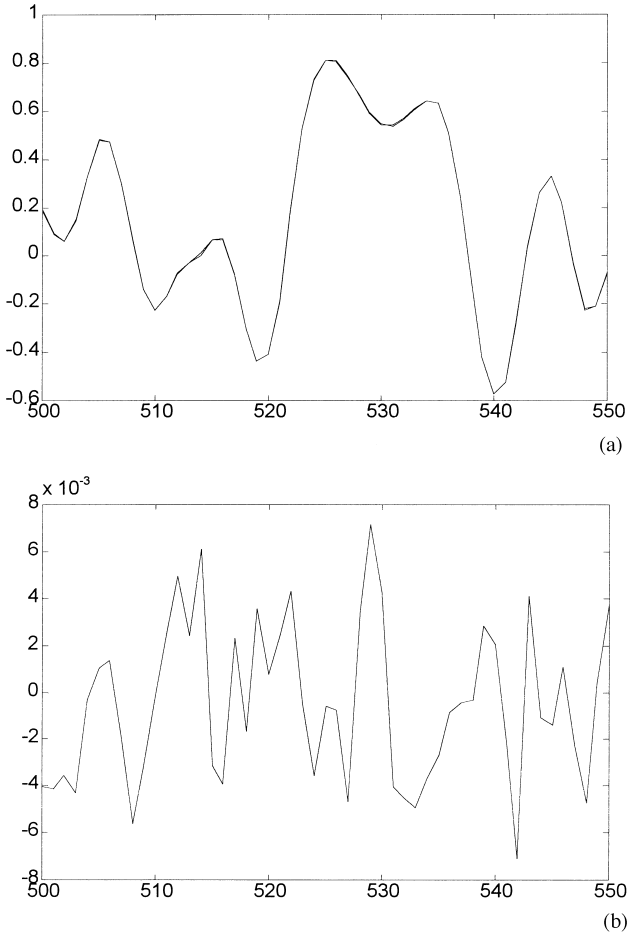

Fig. 7. (a) Generalization capability of the proposed architecture with five GS adaptive neurons: the dashed line, representing the network's output, overlaps exactly the system's output (solid line) on 50 samples of the test set. (b) Generalization capability of the proposed architecture with five GS adaptive neurons: the plot represents the difference between the output of the Back and Tsoi system and the output of the network.

Table 4
Experiments with the Back and Tsoi system: average generalization error for a network with five spline based neurons

Epochs	$\langle g.e. \rangle$ (dB)
1000	−40.493
1500	−41.192
2000	−41.531

3.3. Noisy characters recognition

The last experiment has been carried out in order to test the sensitivity of the Δx parameter with respect to the generalization (or bias vs. variance) performances of the ASNN compared with the standard MLP.

The neural network training set consists in ten noisy characters (the digits 0–9) defined as a 7×7 pixels matrix while the target is the 4-bit binary code of the respective input ((0)−0000,(1)−0001,...,(9)−1001). The noise is obtained randomly flipping 10% of the input pixels (this class of noise is usually called *salt and pepper* noise). For the learning phase we use 100 noisy realization for each character, so in total we have 1000 noisy patterns.

The simulations are performed by training several ASNN of different size and standard MLPs using (about) the same number of adaptable parameters; so at the end of each training phase a similar number of parameters are effectively adapted using, also, a close CPU time for both sort of networks.

Using the same notation of the previous experiments, the networks are denoted as N_y for standard MLP and Sx_y for the ASNN. The symbol y represents the number of hidden neurons, while, for the ASNN, x is related to the Δx value. MLP networks with 20 and 22 hidden neurons are compared with ASNN with 14, 16 and 18 hidden units using several Δx values. Considering all weights and offsets, the MLP with 49 inputs 20 hidden neurons and four output neurons (N_20) has 1084 free parameters, while the N_22 has 1192 free parameters. In the spline networks, for each back-propagation learning iteration, the four spline activation function control points are also adapted, thus, respect to the standard MLP, we must add four adaptable parameters for each neuron.

In order to have statistical information each experiment is repeated five times using different weights initialization.

Table 5
Generalization performance using networks with closed number of free parameters: 1084 for the standard MLP and 1064 for the ASNN; for various Δx values. The learning stop criterion is: MSE = −20 dB to −26 dB. $\mu_w = 0.005$, $\mu_q = 0.005$. Salt and pepper noise = 10%. The training is performed using 100 fixed noise samples for each character

Net	Δx	−20 dB			−22 dB			−24 dB			−26 dB		
		$\langle l.e. \rangle \times 10^3$	MSE $\langle g.e. \rangle$	$\langle$hr%$\rangle$	$\langle l.e. \rangle$	MSE $\langle g.e. \rangle$	$\langle$hr%$\rangle$	$\langle l.e. \rangle \times 10^3$	$\langle g.e. \rangle$	$\langle$hr%$\rangle$	$\langle l.e. \rangle \times 10^3$	MSE $\langle g.e. \rangle$	$\langle$hr%$\rangle$
N_20	–	228	−17.5	93.47	346	−18.2	94.29	520	−18.7	94.80	726	−19.0	95.12
s1_18	0.4	102	−14.9	87.41	142	−15.1	88.21	198	−15.2	88.34	290	−14.9	87.31
s2_18	0.8	66	−17.7	94.29	86	−17.94	94.61	108	−18.1	94.80	136	−18.3	94.95
s3_18	1.2	60	−18.6	95.65	70	−18.8	95.83	88	−19.0	95.93	116	−19.2	96.07
s4_18	1.6	60	−18.8	95.81	70	−19.0	95.97	90	−19.2	96.14	116	−19.3	96.14
s5_18	2.0	70	−18.2	95.91	94	−18.6	96.20	136	−18.9	96.33	210	−19.1	96.37

The result are reported in terms of both standard deviation (SD) and average of the MSE value ($\langle$MSE$\rangle$).

We train all the networks until the $\langle$MSE$\rangle$ (averaged over the last 100 epochs) reaches the value of −25 dB while, during the learning phase, different realizations of salt and pepper noise are added to the input patterns. The percentage of flipped pixels is evaluated considering the entire epoch such that the inverted pixels number for each character can be different for each training step.

The generalization performance are evaluated in terms of generalization error $\langle g.e. \rangle$ and in terms of correct character classification percentage (hit-rate), computed over 1000 forward phase epochs (10 000 characters) where different realizations of salt and pepper noise are used.

The results are reported in Tables 5 and 6. As we can see from Table 5 the generalization performance of the ASNN are better in all the cases. As pointed out above, the choice of the Δx value is not critical. In fact, varying Δx from 0.6 to 2.0, close generalization performances are obtained. We can note also that, to reach the specific learning stop criterion, we need less learning iterations; so the network is trained with a small number of noisy input realizations but with very little degradation of the generalization properties. This propriety is very appreciable in real world problems when the noisy training set is available in a small number of examples.

All training epochs are performed using the weights learning rate $\mu_w = 0.005$, both for standard MLP and ASNN, and using the same value for the spline control points learning rate μ_q. The μ_w and μ_q are empirically determined as optimal values for this class of problem considering the speed of the convergence and the asymptotic error.

4. Conclusions

This paper has presented the most important theoretical properties of a new architecture based on adaptive activation functions, realized through Catmull–Rom cubic splines. The use of cubic splines in single hidden layer structures can be justified in the framework of regularization theory when introducing additive models, but we have to face with two main drawbacks: first of all we build a model made of a sum of just n functions, where n is the dimension of the input space, and this does not guarantee universal approximation; then we are obliged to use a lot of kernels $|x|^3$, which impose a very high hardware complexity. On the contrary, we propose to add, if needed, some more dimensions and to realize the whole cubic spline, working on a particular direction, in a single neuron: to make the implementation easier we do not use the exact form imposed by the theory, but a sub-optimal solution made of local cubic tracts with the same spacing Δx. This parameter, whose choice is not critical above a certain threshold, as reported by experimental results, seems to control the smoothness of the activation functions and the generalization ability. We also pointed out that the mechanism of adaptation of the activation functions works only on local tracts, so that if a segment on the x-axis is never involved by any input data, its four control points will never be updated. This fact, together with the reduction in the network's size due to the augmented expressive power of each neuron, is the reason for a rather low number of free parameters respect to sigmoidal networks (especially for input spaces with high dimensionality). Besides, the local character of the adaptation is an application of the principle of minimal disturbance, since a new input data does not modify the whole

Table 6
Generalization performance using different size networks (Δx fixed for the ASNN). Training MSE = −26 dB

Net	Δx	No. of adapt. param.	$\langle$l.e.$\rangle \times 10^3$ (SD)	$\langle g.e. \rangle$ (SD)	$\langle$No. of wrong patterns$\rangle$	Hit-rate % (SD)
N_20	–	1084	726.00 (8.29)	−19.05 (0.27)	487.60 (35.62)	95.12 (0.36)
N_22	–	1192	766.00 (70.26)	−19.30 (0.27)	441.80 (27.57)	95.58 (0.28)
s4_14	1.6	832	120.00 (6.32)	−18.70 (0.37)	438.00 (40.53)	95.62 (0.41)
s4_16	1.6	948	130.00 (8.94)	−19.01 (0.34)	421.80 (37.72)	95.78 (0.38)
s4_18	1.6	1064	116.00 (4.10)	−19.31 (0.25)	386.20 (23.99)	96.14 (0.24)

shape of the neuron's output, but only a well-defined portion.

It is also important to notice that the way the nonlinearity tracks the target improves learning speed and convergence properties, which means that we need fewer learning epochs and that it is more difficult to end the training in very bad local minima: in general, networks with GS neurons having a good behavior on test sets are easier to find, as if the error surface had a more regular aspect. Simulations with simple test functions, with the Back and Tsoi nonlinear system and with a character recognition problem confirm these expectations.

Acknowledgements

The authors would like to acknowledge the anonymous reviewers for they useful suggestions and comments that improved this manuscript. This work was supported in part by *Ministero dell'Università e della Ricerca Scientifica e Tecnologica* (MURST) of Italy.

Appendix A

Let us now explain the learning algorithm, based on the well known backpropagation, extending the use of spline based activation functions to architectures with more than just one hidden layer.

Considering L total layers and indicating each of them with the index l, $l = 1,...,M$, we can rewrite Eq. (13) as

$$z_k^{(l)} = \frac{s_k^{(l)}}{\Delta x} + \frac{P-2}{2} \tag{A.1}$$

$$a_k^{(l)} = \lfloor z_k^{(l)} \rfloor,$$

$$a_k^{(l)} = z_k^{(l)} - a_k^{(l)}$$

where k is the index of the kth neuron of the layer. Being p the learning step, the adaptation follows the usual rules.

$$w_{kj}^{(l)}[p+1] = w_{kj}^{(l)}[p] + \Delta w_{kj}^{(l)}[p] \tag{A.2}$$

$$q_{k,(a_k^{(l)}+m)}^{(l)}[p+1] = q_{k,(a_k^{(l)}+m)}^{(l)}[p] + \Delta q_{k,(a_k^{(l)}+m)}^{(l)}[p]$$
$$m = 0, 1, 2, 3$$

Referring to Fig. 8, we put $x_k^{(l)} = F_{k,a_k^{(l)}}^{(l)}(u_k^{(l)})$, which is a third order polynomial; if the output is multidimensional and the kth component of the target vector is t_k, we can write, omitting the time p for the sake of readibility

$$e_k^{(l)} = \begin{cases} t_k - x_k^{(l)} & l = M \\ \sum_{m=1}^{N_{l+1}} \delta_m^{(l+1)} w_{mk}^{(l+1)} & l = M-1, \cdots, 1 \end{cases} \tag{A.3}$$

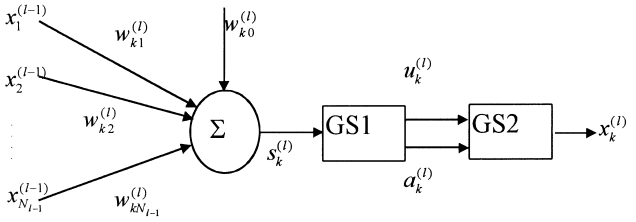

Fig. 8. Schematic structure of the GS neuron: GS1 computes the parameters $u_k^{(l)}$ and $a_k^{(l)}$, while GS2 computes the neuron's output through the spline patch determined by GS1.

$$\delta_k^{(l)} = e_k^{(l)} \left(\left. \frac{dF_{k,a_k^{(l)}}^{(l)}(u)}{du} \right|_{u=u_k^{(l)}} \right) \frac{1}{\Delta x},$$

$$\Delta w_{kj}^{(l)} = \mu_w \delta_k^{(l)} x_j^{(l-1)}$$

with $0 \le k \le N_l$ and $0 \le j \le N_{l-1}$. If we indicate with $c_{k,m}^{(l)}(.)$ the mth column of the matrix in Eq. (10), the adaptation of the control points is ruled by

$$\Delta q_{k,(a_k^{(l)}+m)}^{(l)} = \mu_q e_k^{(l)} \left(\frac{\partial F_{k,a_k^{(l)}}^{(l)}}{\partial q_{k,(a_k^{(l)}+m)}^{(l)}} \right) = \mu_q e_k^{(l)} c_{k,m}^{(l)}(u_k^{(l)}). \tag{A.4}$$

The adaptation rates are μ_w for the connection weights and bias and μ_q for the control points. The control point with index 0,1,$P-1$ and P are fixed.

Appendix B

The cubic spline Eq. (8) obtained through regularization theory can be written in the ith span, where $\alpha_{i+1j} \le \mathbf{w}_j \mathbf{x} = \tilde{x} \le \alpha_{i+2j}$, , as

$$\sum_{h=1}^{N} c_h |w_j x - \alpha_{hj}|^3 = c_1(\tilde{x} - \alpha_{1j})^3 + \cdots + c_i(\tilde{x} - \alpha_{ij})^3 + c_{i+1}(\tilde{x} - \alpha_{i+1j})^3 - c_{i+2}(\tilde{x} - \alpha_{i+2j})^3 - c_{i+3}(\tilde{x} - \alpha_{i+3j})^3 - \cdots - c_N(\tilde{x} - \alpha_{Nj})^3. \tag{B.1}$$

Collecting the terms of equal degree we obtain the simpler form

$$\sum_{h=1}^{N} c_h |w_j x - \alpha_{hj}|^3 = A\tilde{x}^3 + B\tilde{x}^3 + C\tilde{x} + D \tag{B.2}$$

coefficients A, B, C, and D depend on the whole set $c_1,...,c_N$.

We can reproduce the span in the interval $[\alpha_{i+1j},\alpha_{i+2j}]$ by a Catmull–Rom cubic spline: we just need four control points $\mathbf{Q}_{ij1}$, $\mathbf{Q}_{ij2}$, $\mathbf{Q}_{ij3}$ and $\mathbf{Q}_{ij4}$: $\mathbf{Q}_{ij2}$ and $\mathbf{Q}_{ij3}$ lie on the curve while $\mathbf{Q}_{ij1}$ and $\mathbf{Q}_{ij4}$ are only dummy points. Equally spacing the x-coordinates of $\Delta x_{ij} = \alpha_{i+2j} - \alpha_{i+1j}$ and putting $q_{xij2} = \alpha_{i+1j}$, $q_{xij3} = \alpha_{i+2j}$, we easily calculate the local coordinate u by Eq. (11). Imposing the equality

between Eq. (B.2) and the F_y of the Catmull–Rom

$$\frac{1}{2}\left(-q_{yij1}+3q_{yij2}-3q_{yij3}+q_{yij4}\right)\left(\frac{\tilde{x}-\alpha_{i+1j}}{\Delta x_{ij}}\right)^3 + \frac{1}{2}\left(2q_{yij1}-5q_{yij2}+4q_{yij3}-q_{yij4}\right)\left(\frac{\tilde{x}-\alpha_{i+1j}}{\Delta x_{ij}}\right)^2 + \frac{1}{2}\left(-q_{yij1}+q_{yij3}\right)\left(\frac{\tilde{x}-\alpha_{i+1j}}{\Delta x_{ij}}\right)+q_{yij2} = A\tilde{x}^3+B\tilde{x}^2+C\tilde{x}+D \quad \text{(B.3)}$$

we obtain a linear system of four equations where the unknowns are the *y*-coordinates of the points **Q**. These calculations show that a Catmull–Rom spline with properly chosen control points is equivalent to the optimal cubic spline Eq. (B.1) everywhere between α_{1j} and α_{Nj} (outside this interval we prefer to use a bounded function instead of a polynomial). Of course there's no use in ''translating'' Eq. (B.1) in a structure of similar or even greater complexity: the strategy we adopted leads to a much more simplified form that, although sub-optimal from the point of view of regularization theory, is still able to obtain a good performance thanks to its adaptation capabilty.

References

Amari, S. (1993). A universal theorem on learning curves. *Neural Networks, 6*, 161–166.

Amari, S., & Murata, N. (1993). Statistical theory of learning curves under entropic loss criterion. *Neural Computation, 5*, 140–153.

Back, A.D., & Tsoi, A.C. (1993). A simplified gradient algorithm for IIR synapse Multilayer Perceptron. *Neural Networks, 5*, 456–462.

Campolucci, P., Capparelli, F., Guarnieri, S., Piazza, F., & Uncini, A., Neural networks with adaptive spline activation function. *Proceedings of IEEE MELECON 96*, Bari Italy, pp. 1442–1445.

Chen, C. T., & Chang, W. D. (1996). A feedforward neural network with function shape autotuning. *Neural Networks, 9*, 627–641.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, New York: Springer Verlag, pp. 303–314.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*, 1–58.

Girosi, F., Jones, M., & Poggio, T. (1995). Regularization theory and neural networks architectures. *Neural Computation, 7*, 219–269.

Guarnieri, S., Piazza, F., & Uncini, A. (1995). Multilayer neural networks with adaptive spline-based activation functions. *Proceedings of the International Neural Network Society Annual Meeting WCNN 95*, Washington D.C., USA, pp. I695–I699.

Guyon, I., Vapnik, V., Boser, B., Bottou, L., & Solla, S. A. (1992). Structural Risk Minimization for Character Recognition. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, eds, *Advances in Neural Information Processing Systems, 4*, Morgan Kauffmann, pp. 471–479.

Hassoun, M. H. (1996). *Fundamentals of Artificial Neural Networks*. Cambridge, MA: MIT.

Holden, S.B., & Rayner, P.J.W. (1995). Generalization and PAC learning: some new results for the class of generalized single-layer networks. *IEEE Transactions on Neural Networks, 6 (2)*, 368–380.

Hwang, J.-N., Lay, S.-R., Maechler, M., Martin, R.D., & Schimert, J. (1994). Regression modeling in back-propagation and projection pursuit learning. *IEEE Transactions on Neural Networks, 5 (2)*, 342–353.

Moody, J.E. (1992). The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In J.E. Moody, S.J. Hanson, & R.P. Lippmann, eds, *Advances in Neural Information Processing Systems, 4*, Morgan Kauffmann, pp. 847–854.

Moody, J.E., & Utans, J. (1994). Architecture selection startegies for neural networks: application to corporate bond rating prediction. In A.N. Refenes, ed., *Neural Networks in the Capital Markets*, John Wiley.

Niyogi, P., & Girosi, F. (1996). On the relationship between generalization error hypothesis complexity and sample complexity for radial basis functions. *Neural Computation, 8*, 819–842.

Piazza, F., Smerilli, S., Uncini, A., Griffo, M., & Zunino, R. (1996). Fast spline neural networks for image compression. In *WIRN96*, Vietri sul Mare, Italy.

Piazza, F., Uncini, A., & Zenobi, M. (1992). Artificial neural networks with adaptive polynomial activation function. *Proc. of the IJCNN*, vol. 2, Bejing, China, pp. 343–349.

Piazza, F., Uncini, A., & Zenobi, M. (1993). Neural networks with digital LUT activation function, *Proc. of the IJCNN*, vol. 2, Nagoya, Japan, pp. 1401–1404.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE, 78 (9)*, 1481–1497.

Poggio, T., & Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science, 247*, 978–982.

Reed, R., Marks II, R.J., & Oh, O. (1995). Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitters. *IEEE Transactions on Neural Networks, 6 (3)*, 529–538.

Stinchcombe, M., & White, H. (1989). Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. *Proc. of the IJCNN*, vol. 1, Washington D.C., USA, pp. 613–617.

Widrow, B., & Lehr, M.A. (1990). 30 Years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE, 78 (9)*, 1415–1442.

Wolpert, D.H. (1995). On bias plus variance. Technical Report SFI TR 95-08-074, Santa Fe Institute.