Multilayer Feedforward Networks with Adaptive Spline Activation Function

Stefano Guarnieri, Francesco Piazza, *Member, IEEE*, and Aurelio Uncini, *Member, IEEE*

***Abstract*— In this paper, a new adaptive spline activation function neural network (ASNN) is presented. Due to the ASNN's high representation capabilities, networks with a small number of interconnections can be trained to solve both pattern recognition and data processing real-time problems. The main idea is to use a Catmull–Rom cubic spline as the neuron's activation function, which ensures a simple structure suitable for both software and hardware implementation. Experimental results demonstrate improvements in terms of generalization capability and of learning speed in both pattern recognition and data processing tasks.**

***Index Terms*— Adaptive activation functions, function shape autotuning, generalization, generalized sigmoidal functions, multilayer perceptron, neural networks, spline neural networks.**

I. Introduction

In both hardware and software neural network (NN) implementations, the complexity, both structural, in terms of interconnections, and computational, in terms of the number of multiplications, is a bottleneck for many real-time applications. However, it is known that under certain regularity conditions, the NN representation capabilities depend on the number of free parameters, whatever the structure of the network [14]. Hence, adaptable activation functions can reduce the number of interconnections and therefore the overall network complexity, since they now contain free parameters. In digital NN implementation, the computational complexity of the forward phase can be further reduced when the activation function is implemented through a lookup table (LUT) [15]. The LUT access time is obviously independent from the function shape.

The classical neuron model, proposed by McCulloch and Pitts in 1943 [1], is composed of a linear combiner followed by a nonlinear function (activation function) with hard-limiting characteristics. More recently, in order to develop gradient-based learning algorithms, such hard-limiters are substituted with nonlinear differentiable functions [2]. It is well known that the network behavior, as that shown by the multilayer perceptron (MLP), greatly depends on the shape of these activation functions. Sigmoidal functions are the most common in NN applications [2], [3], however, other kinds of functions, at times depending on some free parameters, are used to improve the NN's representation capabilities.

Manuscript received July 15, 1997; revised August 17, 1998 and March 1, 1999. This work was supported in part by *Ministero dell'Università e della Ricerca Scientifica e Tecnologica* (MURST) of Italy.

The authors are with the Dipartimento di Elettronica e Automatica, Università di Ancona, Italy, 60131 Ancona, Italy.

Publisher Item Identifier S 1045-9227(99)03999-5.

In [4], the authors proposed a MLP with adaptive-slope sigmoidal activation function. In [5], a more general approach is put forward: an adaptive generalized hyperbolic tangent function with two free parameters, the slope and the saturation level (or gain), is suggested. Moreover, NN's universal approximation capabilities are guaranteed for a large class of activation functions [6]–[8] and the behavior of different classes of activation functions have been studied in depth by several authors [7]–[12].

In [16], the authors proposed a class of NN's, based on adaptive polynomial activation functions, called adaptive polynomial NN (APNN). They demonstrated that such networks have similar behavior to the conventional Volterra's polynomial classificators or filters. As some of the free parameters are now related to the polynomial activation functions, the APNN can be used to obtain networks with reduced structural complexity. Although extremely efficient for a large class of applications, the APNN's are not universal approximators [11]. Furthermore, during the learning phase, APNN could have problems due to local minima and, in the case of high-degree polynomials, numerical instability could arise (often due to the presence of high narrow peaks in the polynomial functions).

In [17] a different approach to obtain an adaptive activation function for digital NN's was proposed. Here, the activation function itself is realized by the LUT, and the derivatives, needed by any gradient-based learning algorithm, are computed as a numerical approximation by finite differences. The LUT adaptation algorithm can be implemented very efficiently in hardware. The experiments reported in [17], using the adaptable LUT neural networks (ALNN's), show that some canonical problems can be solved, notably reducing the network dimensions. For example, it is possible to learn and solve the classical multibit parity problem with only one neuron. The ALNN's, however, present a large number of adaptable parameters; in fact, each LUT element represents a free parameter. As a consequence, the learning phase can be difficult. Moreover, when the network is used to solve a data processing problem (e.g., signal processing, dynamical systems identification, etc.) the quantized characteristics of the activation function can reduce the network performance.

In [24], an approach based on Hermite polynomials as substitutes for the so-called super-smoother in the projection pursuit learning (PPL) is proposed. Although the authors present a constructive learning algorithm, while in this paper we retain classical gradient-based learning, 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.

1045–9227/99$10.00 © 1999 IEEE

The main idea of this paper is to present a new architecture based on adaptive *spline* activation functions. These adaptive spline-based NN's (ASNN's) are designed using a neuron, called generalized sigmoidal (GS-neuron), containing an adaptive parametric spline activation function. The basic network scheme is therefore very similar to classical MLP structures [3], but with improved nonlinear adaptive activation functions.

These functions have several interesting features: they 1) are easy to adapt; 2) retain the squashing property of the sigmoid; 3) have the necessary smoothing characteristics; and 4) are easy to implement both in hardware and in software. The multilayer networks built with such neurons are still universal approximators and usually have a smaller structural complexity, maintaining good generalization capabilities.

The paper investigates the representation properties of spline-based neurons, and analyzes the advantages of this approach in some applications, both in pattern recognition and data processing domains. In particular, we show that among the available spline curves, the cubic Catmull–Rom (CR) spline [19], [20] is suitable for implementing an adaptive activation function with some interesting features, due to its local interpolation and regularization characteristics.

In Section II, the new ASNN using cubic splines is presented. In Section III, a backpropagation learning algorithm is derived. Finally, in Section IV, several experimental results, both on pattern recognition and data processing problems are reported, in order to show the performance of the proposed model.

II. The Adaptive Spline Neural Networks

A. Basic Architectural Considerations

The problem addressed in this section is to find a nonlinear adaptive function (or a curve), suitable for implementing an activation function, that: 1) satisfies the *boundedness* constraint defined for activation functions [6]–[12]: 2) is able to retain the MLP universal approximation property; 3) is flexible enough to modify its shape by adapting a small number of parameters.

Most of the functions currently used lack such a flexibility. If an adaptive polynomial is used, a modification of a single parameter can lead to a change in the whole shape. The other adaptive functions proposed in the literature, which satisfy the universal approximation requirements (as a sigmoid with a adaptable slope or/and gain [4], [5]), are such that a slight modification in a single point can influence the overall shape of the function. This global adaptation can sometimes bring oscillatory behavior, thus cancelling the previously learned information: the stability of the learning process is deemed to be compromised.

A LUT-based activation function can overcome this problem, localizing the adaptation process so that the shape of the nonlinear mapping changes locally after each adaptation step. In fact, each step now affects only a small set of contiguous values in the LUT, preserving most of the adaptation performed in the previous learning. Only after a certain number of these steps, will the shape of the activation function reflect the information represented by the data set.

The LUT values can therefore be seen as the curve sample points to be adapted. As previously stated, the direct use of a LUT activation function can lead to a huge number of free parameters also in small networks. In order to reduce this number, in [17] the authors limit the dimension of the LUT and use a suitable interpolation scheme.

The choice of an interpolation scheme is not an obvious one, in spite of the fact that even piecewise linear activation functions retain the universal approximation property [11]. In fact, a wrong choice of the interpolation scheme can lead to problems in the development of the learning algorithm: in [17], it was found that a zero-order interpolation (like a sample-and-hold system) suffers from numerical instability, because the derivative, evaluated as the difference of sequential values, is not a continuous approximation.

A good interpolation scheme should guarantee a continuous first derivative, as well as the capability to locally adapt the curve: such properties are exhibited by the so-called piecewise polynomial spline interpolation schemes [18], that we use in this paper.

Let a LUT be defined as a $N+1$ point array, $\{Q_0, \cdots, Q_N\}$, where each point is represented by the x and y coordinates $Q_i = [q_{x,i} \;\; q_{y,i}]^T$, where T is the transpose operator. The LUT points are also constrained to be such that

$$q_{x,0} < q_{x,1} < \cdots < q_{x,N}$$

to avoid loops (reverse ordering of abscissas) or multiple output values for a single abscissa (overlapping abscissas). This constraint is necessary, since it avoids representing non-injective functions.

A planar spline curve is a two-dimensional vector, whose components are piecewise polynomial univariate functions of the same degree: its mathematical formulation ensures both its continuity and the existence of its derivatives, along the curve and in correspondence to the joining points between the various curve spans. Given a LUT as defined above, a general spline expression for that curve would be

$$F(u) = [F_x(u) \quad F_y(u)]^T = \mathop{\mathbf{C}}_{i=0}^{N-3} F_i(u) \tag{1}$$

where $\mathbf{C}$ is the concatenation operator and $F_i(u)$ of ith curve span (or patch). The indices of the $\mathbf{C}$ operator in (1) are valid only for cubic polynomials: the choice of using cubic polynomials was made because of the tradeoff between the requested properties and the computational complexity.

The parameter u has the property of being *local* and its domain is $0 \leq u \leq 1$ for every curve span. Hence, there must be a unique mapping that allows us to calculate the *local* parameter u, as well as the proper curve span i, from the abscissa *global* parameter. In this way, we can represent any point lying on the spline curve $F(u)$ as a point belonging to the single $F_i(u)$ curve span. It follows (see [19]) that the ith curve span can be described as follows:

$$F_i(u) = [F_{xi}(u) \quad F_{yi}(u)]^T = \sum_{j=0}^{3} Q_{i+j} C_j(u) \tag{2}$$

where $C_j(u)$ are the spline polynomials. As in (1), each coordinate is described by a univariate function, namely a cubic polynomial of the variable u.

B. The GS Neuron

Let s be the linear combiner output of a neuron, usually called "activation;" we have to make the dependence between s and $F_i(u)$ explicit. There is a direct link between s and one of the two components (a cubic polynomial function), namely

$$s = F_{xi}(u). \tag{3}$$

Equation (3) plays a key role in finding a suitable activation function architecture, as it is the main bottleneck of the overall structure: in fact, it is the only link between the linear combiner output s and nonlinear neuron output $F_{yi}(u)$, which is given in term of the $i \in [0, N]$ value (which represents the LUT index) and the $u \in [0, 1]$ value (which represents the offset of the ith curve span). Let y be the neuron output, we have to introduce a two-step procedure to calculate y of a single neuron, given the output s of the linear combiner:

- calculate u and i from s by inverting (3);
- substitute these values of u and i in $y = F_{yi}(u)$.

Equation (2) resolves the link between the curve and the sample points Q_i which are called *control points* in the literature, as they control the shape of the curve. The cubic polynomial functions $C_j(u)$, called *spline basis functions* or *blending functions*, characterize the way the curve moves along the path made up by the control points. The curve can interpolate or just approximate its control points, depending on which blending function set we rely on (see for example [19]).

In our case, we prefer an interpolation scheme, due to its local characteristics, which avoids the oscillatory behavior of the global adaptation of the approximation scheme. Actually, there are few splines that interpolate their control points; one that also has a very low computational overhead is the so-called *CR cubic spline basis* [20], which is described by the following polynomials [with reference to (2)]:

$$\begin{aligned} C_0(u) &= \frac{1}{2}(-u^3 + 2u^2 - u) \\ C_1(u) &= \frac{1}{2}(3u^3 - 5u^2 + 2) \\ C_2(u) &= \frac{1}{2}(-3u^3 + 4u^2 + u) \\ C_3(u) &= \frac{1}{2}(u^3 - u^2). \end{aligned} \tag{4}$$

A quick inspection of (4) shows that all the multiplications, except for the powers of the parameter u, are by integer coefficients and that they should be easily implemented in hardware (just one or two shifts or sum-and-shift operations). This characteristics of the CR spline is important, as it simplifies the structure of the nonlinear block. A common expression of (2), is the following matricial notation:

$$F_i(u) = [u^3 \quad u^2 \quad u \quad 1]\frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \tag{5}$$

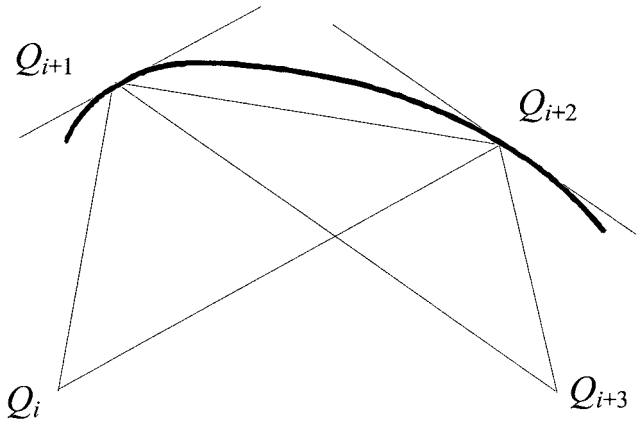


Fig. 1. Catmull–Rom spline curve span.

which explicits the actual array structure of a curve span: a *parameter vector*, a *basis matrix* and a *control point vector*, combined using row by column multiplications. Fig. 1 shows the graphical representation of the single CR ith curve span.

Deriving (5) with respect to u, it holds that

$$\frac{\partial F_i(u)}{\partial u} = [3u^2 \quad 2u \quad 1]\frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} Q_i \\ Q_{i+1} \\ Q_{i+2} \\ Q_{i+3} \end{bmatrix} \tag{6}$$

where for $u = 0$ we have $\frac{\partial F_i(0)}{\partial u} = \frac{1}{2}(-Q_i + Q_{i+2})$ and for $u = 1$ we have $\frac{\partial F_i(1)}{\partial u} = \frac{1}{2}(-Q_{i+1} + Q_{i+3})$. This means that the curve tangency lines, at the points Q_{i+1} and Q_{i+2}, are parallel to the straight lines passing through the points Q_i and Q_{i+2} and through the points Q_{i+1} and Q_{i+3}, respectively (see Fig. 1). The continuity of the derivative is also useful for the definition of gradient-based learning algorithms.

The general result of (5) can be expressed as a pair of cubic polynomials (where a, b, c and d are appropriate constants)

$$\begin{aligned} F_{xi}(u) &= a_{xi}u^3 + b_{xi}u^2 + c_{xi}u + d_{xi} \\ F_{yi}(u) &= a_{yi}u^3 + b_{yi}u^2 + c_{yi}u + d_{yi}. \end{aligned} \tag{7}$$

Fig. 2 shows the polynomials $F_{xi}(u)$ and $F_{yi}(u)$ used for implementing the activation function $y = f(s)$.

In (3), the inversion of the first polynomial in (7) would be needed to get the parameters u and i, which is in turn necessary to generate the neuron's output. We could exploit the formulas for the solution of third-order equations (algebraically or iteratively), but a serious overhead in the calculations would be introduced. Moreover, this approach gives no valid answer to the ordering problem that affects the abscissa control points.

There is, however, a regularization property common to most of the polynomial spline basis set (CR included) called *variation diminishing property*, which ensures the absence of unwanted oscillations of the curve between two consecutive control points, as well as the exact representation of linear segments. This second statement is particularly important, as it suggests a simple possible solution to the inversion problem (3): if we uniformly sample the abscissas along the x-axis,

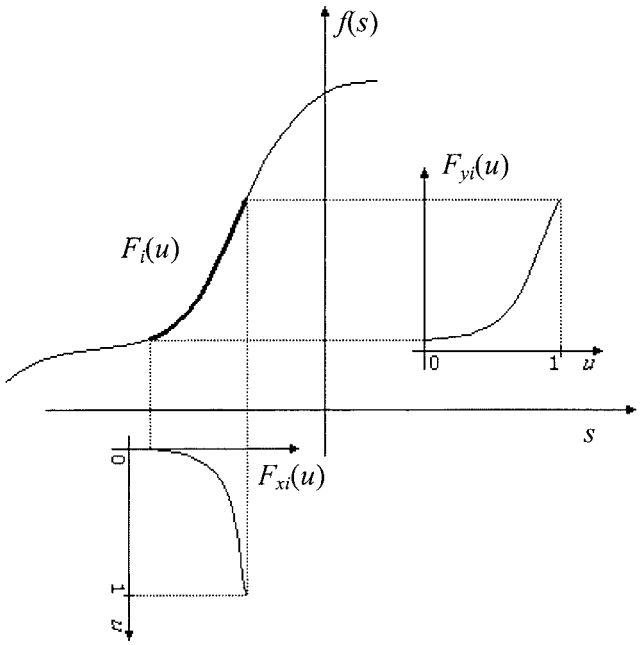


Fig. 2. Single curve span decomposed in its two components.

then the cubic polynomial $F_{xi}(u)$ becomes a *first degree* polynomial. This approach ensures both a fast computation of the local parameters u and i, and the monotone ordering of the abscissas, in the case when they are kept fixed.

In this case, let us consider a fixed sample step $\Delta x = q_{x,i+1} - q_{x,i}$: after substituting the proper values of the abscissa control points in (5), the function $F_{xi}(u)$ has the following form:

$$F_{xi}(u) = [u^3 \quad u^2 \quad u \quad 1]\frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} q_{x,i} \\ q_{x,i} + \Delta x \\ q_{x,i} + 2\Delta x \\ q_{x,i} + 3\Delta x \end{bmatrix}. \tag{8}$$

The solution to this equation is a linear mapping

$$F_{xi}(u) = u\Delta x + q_{x,i} + \Delta x = u\Delta x + q_{x,i+1} \tag{9}$$

in which all the parameters are known. Expression (9) is very fast to compute, and it leads to a negligible computational overhead. Moreover, it is not necessary to store the $q_{x,i}$ table, as these values can now be determined algorithmically (more on this later).

The (7) can now be written as

$$\begin{aligned} F_{xi}(u) &= s = c_{xi}u + d_{xi} \\ F_{yi}(u) &= y = a_{yi}u^3 + b_{yi}u^2 + c_{yi}u + d_{yi}. \end{aligned} \tag{10}$$

Operating the inversion $u = F_{xi}^{-1}(s)$, the output y can be computed as a cubic polynomial which expresses a direct relation between the input s and the output y of the CR-based nonlinear block

$$\begin{aligned} y = F_{yi}(s) &= \frac{a_{yi}}{c_{xi}^3}s^3 + \left(\frac{b_{yi}}{c_{xi}^2} - \frac{3a_{yi}d_{xi}}{c_{xi}^3}\right)s^2 \\ &+ \left(\frac{3a_{yi}d_{xi}^2}{c_{xi}^3} - \frac{2b_{yi}d_{xi}}{c_{xi}^2} + \frac{c_{yi}}{c_{xi}}\right)s \\ &+ \left(\frac{b_{yi}d_{xi}^2}{c_{xi}^2} - \frac{a_{yi}d_{xi}^3}{c_{xi}^3} - \frac{c_{yi}d_{xi}}{c_{xi}} + d_{yi}\right). \end{aligned} \tag{11}$$

The geometric tangent continuity of the CR curve demonstrated above is clearly valid here, but we are now dealing with *polynomial function spans*, and not curve spans: this leads us to the conclusion that the continuity of the first derivative of the single function span (11) is preserved along the complete function.

The expression (11), although simple to understand, cannot be efficiently calculated in this form: it is for the sake of computational efficiency that we will use (5) for the actual computation of $F_{yi}(u)$.

The result found in (9) is then employed to improve the efficiency of the complete structure. Therefore, we constrain the control point abscissas to be equidistant and, most important, not adaptable. Moreover, always for the sake of efficiency, another constraint is imposed on the control points, forcing the sampling interval to be centred on the x-axis origin. It is then possible to represent the abscissa of each point of the activation function using two parameters (the span index i and the local parameter u), without storing the control points abscissas $q_{x,i}$. All we need to know is how many control points the curve has, and the sampling step Δx.

An additional and important constraint is to force the activation function to be a *limiting function*, i.e., to be constant for $s \to \pm\infty$, while maintaining the ability to modify its shape inside these constant values. A practical implementation of this class of functions starts with the logistic functions

$$f(s) = \frac{2c_1}{1 + e^{-c_2 s}} - c_3 \tag{12}$$

where c_1, c_2, and c_3 are suitable constant parameters (for example, $c_1 = 0.5$, $c_2 = 1$, and $c_3 = 0$ represents the standard unsigned sigmoid, while $c_1 = 1$, $c_2 = 1$ and $c_3 = 1$ represents the signed, or bipolar, sigmoid). Performing a uniform sampling of $f(s)$ with $N + 1$ control points in an interval centred on the x-axis origin and then fixing the first two and the last two control point abscissas, constant output values are obtained outside the sampling interval, while the activation function can be freely shaped between $q_{x,1}$ and $q_{x,N-1}$ (see Fig. 3), starting from the sigmoidal shape as an initial condition.

III. Gradient-Based Learning for ASNN

As an extension of the formalism used by Widrow and Lehr [3] for a multilayer neural network with M layer and N_l $(l = 1, \cdots, M)$ neurons per layer, let us define the following quantities:

$x_k^{(l)}$ Output of the kth neuron in the lth layer ($x_k^{(0)}$ are the network inputs and $x_0^{(l)} = 1$);

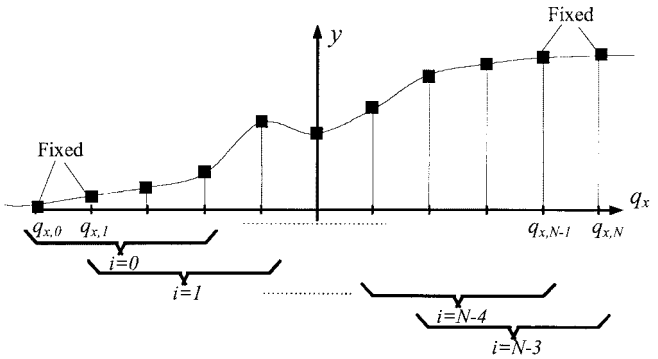


Fig. 3. Control points of the Catmull–Rom spline-based activation function with a fixed step Δx. The extreme points $q_{x,0}$, $q_{x,1}$, and $q_{x,N-1}$, $q_{x,N}$ are fixed.

$w_{kj}^{(l)}$ Weight of the kth neuron in the lth layer with respect to the jth neuron in the previous layer ($w_{k0}^{(l)}$ are the bias terms);

$s_k^{(l)}$ Net output (i.e., the linear combiner output) of the kth neuron in the lth layer;

$N+1$ Number of control points for each neuron in the network;

Δx Sampling step along the x-axis for each activation function;

$i_k^{(l)}$ Curve span index of the activation function for the kth neuron in the lth layer ($0 \le i_k^{(l)} \le N-2$);

$u_k^{(l)}$ Local parameter for the $i_k^{(l)}$th curve span of the kth neuron in the lth layer ($0 \le u_k^{(l)} \le 1$);

$q_{k,n}^{(l)}$ Ordinate of the nth control point of the kth neuron (previously denoted as $q_{y,n}$; the index y does not appear since we assumed y-index implicitly) in the lth layer ($0 \le n \le N$). The control point abscissas $q_{x,n}$ do not appear since we assume the x-axes are uniformly sampled;

$F_{k,i_k^{(l)}}^{(l)}(\cdot)$ $i_k^{(l)}$th spline patch of the activation function for the kth neuron in the lth layer;

$C_{k,m}^{(l)}(\cdot)$ mth CR polynomial (blending function) for the kth neuron in the lth layer ($0 \le m \le 3$).

A. Forward Computation

The result obtained in (9) can now be expressed by the following set of equations, that are intended to be evaluated in strict order ($k = 1, \cdots, N_l$ $l = 1, \cdots, M$)

$$\begin{aligned} z_k^{(l)} &= \frac{s_k^{(l)}}{\Delta x} + \frac{N-2}{2} \\ a_k^{(l)} &= \lfloor z_k^{(l)} \rfloor \\ u_k^{(l)} &= z_k^{(l)} - \lfloor z_k^{(l)} \rfloor \end{aligned} \tag{13}$$

where $\lfloor\ \rfloor$ is the floor operator, and $z_k^{(l)}$ is an internal dummy variable. As an implementation note, the second term in the second equation is needed to force $i_k^{(l)}$ to be always nonnegative. The last three equations solve the inversion problem: they are cheap in terms of computational demand, and provide the two parameters needed to calculate the output. This operation is performed as in (5), that now becomes

$$x_k^{(l)} = F_{k,i_k^{(l)}}^{(l)}\left(u_k^{(l)}\right) = \sum_{m=0}^{3} q_{k,(i_k^{(l)}+m)}^{(l)} C_{k,m}^{(l)}\left(u_k^{(l)}\right). \tag{14}$$

We call the two blocks expressed by (13) and (14) GS1 and GS2, respectively (see Fig. 4).

B. Backward Computation (Learning Phase)

When an iterative learning algorithm is employed, it can be noted that at each iteration all the weights are changed, whereas for each neuron, only the four control points of the involved curve span are updated. This is a consequence of locality of the spline interpolation scheme.

Let t be the iteration index, $d_k[t]$ the desired output, and $E_p[t] = \sum_{k=1}^{N_M} (d_k[t] - x_k^{(M)}[t])^2$ the instantaneous squared output error, related to the pth learning pattern; the weights and control points are adapted following the anti-gradient of the error surface as follows ($l = M, \cdots, 1; k = 1, \cdots, N_l$):

$$w_{kj}^{(l)}[t+1] = w_{kj}^{(l)}[t] - \mu_w \frac{\partial E_p[t]}{\partial w_{kj}^{(l)}[t]} \tag{15}$$

$$q_{k,(i_k^{(l)}+m)}^{(l)}[t+1] = q_{k,(i_k^{(l)}+m)}^{(l)}[t] - \mu_q \frac{\partial E_p[t]}{\partial q_{k,(i_k^{(l)}+m)}^{(l)}[t]} \qquad m = 0, \cdots, 3 \tag{16}$$

where the parameters μ_w and μ_q represent the learning rates for the weights and for the control points, respectively.

For each layer, following the backpropagation approach, we compute the expression

$$e_k^{(l)}[t] = \begin{cases} \left(d_k[t] - x_k^{(l)}[t]\right) & l = M \\ \sum_{p=1}^{N_{l+1}} \delta_p^{(l+1)}[t] w_{pk}^{(l+1)}[t] & l = M-1, \cdots, 1 \end{cases} \tag{17}$$

where the δ's take the values

$$\delta_k^{(l)}[t] = e_k^{(l)}[t] \left(\frac{\partial F_{k,i_k^{(l)}}^{(l)}\left(s_k^{(l)}[t]\right)}{\partial s_k^{(l)}[t]} \right) \tag{18}$$

so that using the chain rule yields

$$\begin{aligned} \frac{\partial F_{k,i_k^{(l)}}^{(l)}\left(s_k^{(l)}[t]\right)}{\partial s_k^{(l)}[t]} &= \frac{\partial F_{k,i_k^{(l)}}^{(l)}\left(s_k^{(l)}[t]\right)}{\partial u_k^{(l)}[t]} \frac{\partial u_k^{(l)}[t]}{\partial s_k^{(l)}[t]} \\ &= \frac{\partial F_{k,i_k^{(l)}}^{(l)}\left(s_k^{(l)}[t]\right)}{\partial u_k^{(l)}[t]} \frac{1}{\Delta x}. \end{aligned} \tag{19}$$

Since the extremal control points $q_{k,0}^{(l)}$, $q_{k,1}^{(l)}$, $q_{k,N-1}^{(l)}$, $q_{k,N}^{(l)}$ are *a priori* fixed, the following constant values of the derivatives are used for them:

$$\begin{aligned} \frac{\partial F_{k,i_k^{(l)}}^{(l)}\left(s_k^{(l)}[t]\right)}{\partial s_k^{(l)}[t]} &= \frac{q_{k,1}^{(l)} - q_{k,0}^{(l)}}{\Delta x} \quad s_k^{(l)} < q_{k,l}^{(l)} \\ \frac{\partial F_{k,j_k^{(l)}}^{(l)}\left(s_k^{(l)}[t]\right)}{\partial s_k^{(l)}[t]} &= \frac{q_{k,N}^{(l)} - q_{k,N-1}^{(l)}}{\Delta x} \quad s_k^{(l)} > q_{k,N-1}^{(l)} \end{aligned} \tag{20}$$

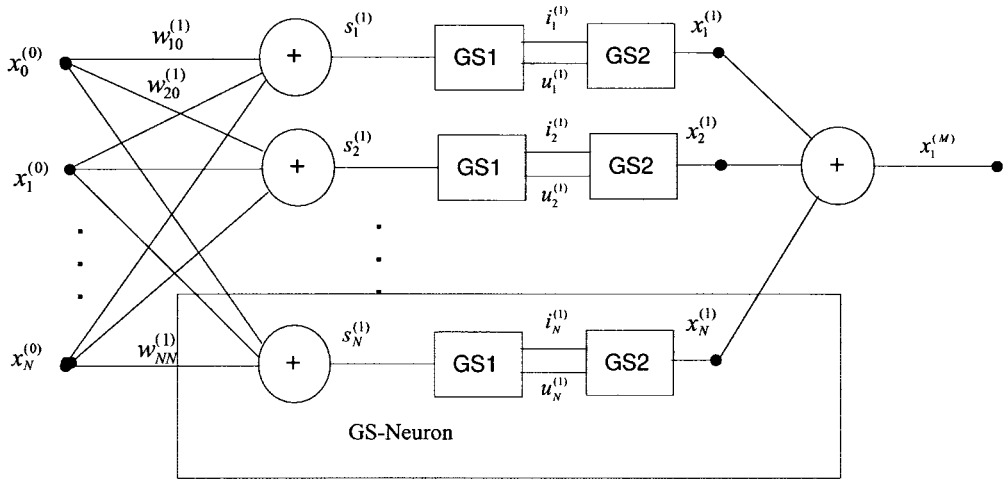


Fig. 4. An example of an adaptive spline neural network (ASNN).

For the other control points we must compute the following derivatives:

$$\frac{\partial E_p[t]}{\partial q^{(l)}_{k,(i^{(l)}_k+m)}[t]} = \frac{\partial E_p[t]}{\partial x^{(l)}_k[t]} \frac{\partial x^{(l)}_k[t]}{\partial q^{(l)}_{k,(i^{(l)}_k+m)}[t]} \tag{21}$$

$$\frac{\partial x^{(l)}_k[t]}{\partial q^{(l)}_{k,(i^{(l)}_k+m)}[t]} = \frac{\partial F^{(l)}_{k,(i^{(l)}_k+m)}\left(s^{(l)}_k[t]\right)}{\partial q^{(l)}_{k,(i^{(l)}_k+m)}[t]} = C^{(l)}_{k,m}\left(u^{(l)}_k[t]\right)$$
$$\text{for } m = 0, \cdots, 3. \tag{22}$$

Considering the weights $w^{(l)}_{k0}$ as the offset values, the (15), (16) can be finally rewritten as $(l = M, \cdots, 1; k = 1, \cdots, N_l)$

$$w^{(l)}_{kj}[t+1] = w^{(l)}_{kj}[t] + \mu_w \delta^{(l)}_k[t] \begin{cases} x^{(l-1)}_j & j \neq 0 \\ 1 & j = 0 \end{cases} \tag{23}$$

$$q^{(l)}_{k,(i^{(l)}_k+m)}[t+1] = q^{(l)}_{k,(i^{(l)}_k+m)}[t] + \mu_q e^{(l)}_k[t] C^{(l)}_{k,m}\left(u^{(l)}_k[t]\right)$$
$$m = 0, \cdots, 3. \tag{24}$$

This algorithm reduces to the standard backpropagation rule [2] when the control points are not adapted (for example by setting the learning rate μ_q to zero).

IV. Experimental Results

ASNN has been tested in several nontoy problems, see, e.g., [21] and [25]. Here we report the results for three common problems usually used in the literature to evaluate NN performance. The first experiment refers to a pattern recognition problem, whereas the second and the third concern data processing problems: chaotic time-series prediction and system identification.

The learning parameters $(\mu_q, \mu_p, \Delta x, N)$ used in all the experiments were chosen in order to optimize the network's performance. However, their setting is not critical; for example, very similar performances were obtained with several Δx values, in the range from 0.5 to 1.0. In all the experiments, the initial weight values were randomly chosen, while the initial spline control points were obtained by sampling a standard sigmoid.

A. Pattern Recognition Experiment

The first experiment is a standard character recognition problem. The training set consists in 64 PC-CGA characters (ASCII codes from 33 to 96) as inputs (7×8 pixel matrix), and the related 7-bit ASCII code as target.

To evaluate the learning and generalization performance of the proposed architecture, several comparisons with standard-MLP were carried out. The runs consist in training different sized ASNN's and standard-MLP's with (about) the same numbers of adaptable parameters.

The ASNN's are denoted as S56_x_1_7, while standard MLP's are denoted as N56_x_2_7, where x_1, x_2 represent the number of neurons in the hidden layer. In our case, in order to have a close number of adaptable parameters for both network families, the parameters x_1 and x_2 belong to the following sets of values: $x_1 = [27, 37, 46]$ and $x_2 = [30, 40, 50]$. As an example, the network S56_46_7 is an ASNN containing 46 neurons in the hidden layer (3163 adaptable parameters), while the network N56_50_7, that is, the closer in terms of number of free parameters, is a MLP with 50 neurons in the hidden layer (3207 adaptable parameters).

The networks are trained using 10 000 epochs of 64 characters each. During the learning phase the input patterns are corrupted by inverting randomly 2% of the pixels (this class of noise is usually called *salt and pepper* noise). The percentage of salt and pepper noise is evaluated considering the entire epoch, such that the number of inverted pixels in each character can be different for each training step.

As values of Δx inside a large range (from 0.5 to 2.0) yield very close generalization performances, all training epochs are carried out using $\Delta x = 1.0$. The weight learning rate is $\mu_w = 0.005$, both for standard MLP and ASNN; the same value is used for the spline control point learning rate μ_q. The parameters μ_w and μ_q were set, on the basis of previous experience with similar problems, to values ensuring high convergence speed and low asymptotic error.

Fig. 5 shows the plot of the average MSE (denoted as $\langle \text{MSE} \rangle$) evaluated in dB as $MSE_{\text{dB}} = 10 \log_{10}(\sum_{p=1}^{N_p} \sum_{k=1}^{N_M} (d_k[t] - x^{(M)}_k[t])^2 / N_p)$, where N_p

TABLE I
CHARACTERS RECOGNITION GENERALIZATION PERFORMANCE. THE TERM $\langle \mathrm{MSE} \rangle$ REPRESENTS THE AVERAGE MEAN SQUARE ERROR, $\langle \mathrm{H.R.} \rangle$ IS THE HIT-RATE AND THE TERM $\langle \mathrm{W.P.} \rangle$ IS THE AVERAGE NUMBER OF WRONG PATTERNS; AVERAGED OVER THREE NETWORKS FOR 1000 FORWARD EPOCHS FOR A TOTAL OF 64000

Network	Num. of adapt. param.	Salt and pepper noise 3% flipped pixels			Salt and pepper noise 5% flipped pixels		
x_1 or x_2	-	<MSE>[dB] (σ)	<H.R.> %	<W.P.>	<MSE>[dB] (σ)	<H. R.> %	<W.P.>
S56_27_7	1871	-20.04 (0.184)	94.17	373.00	-15.85 (0.231)	84.91	965.33
N56_30_7	1927	-19.60 (0.223)	93.65	406.33	-16.00 (0.311)	84.85	969.33
S56_37_7	2551	-21.56 (0.152)	96.15	246.00	-17.24 (0.192)	88.86	712.66
N56_40_7	2567	-20.71 (0.225)	95.29	301.33	-16.87 (0.283)	87.43	804.00
S56_46_7	3163	-22.30 (0.142)	96.51	223.33	-17.73 (0.187)	89.42	676.66
N56_50_7	3207	-21.11 (0.192)	95.58	282.66	-17.26 (0.202)	88.40	742.33

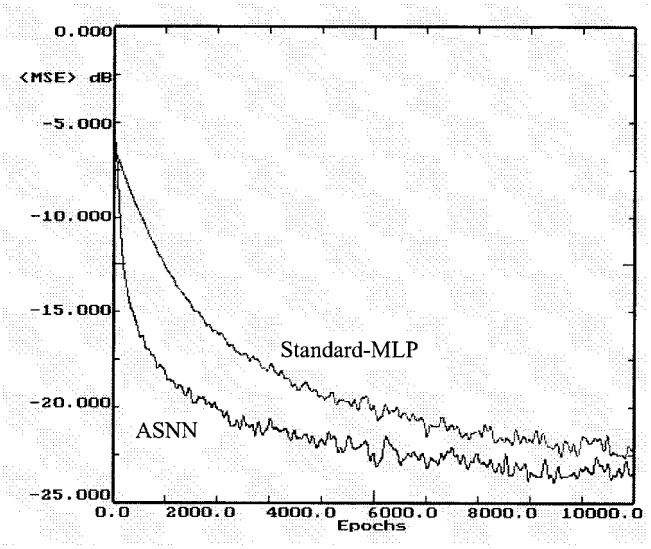


Fig. 5. Averaged learning curve comparison between S56_37_1 and N56_40_1 networks, with 2551 and 2567 adaptable parameters, respectively.

is the number of training patterns ($N_p = 64$, $N_M = 7$ for this experiment), during the learning phase for the networks N56_40_7, containing 2567 free parameters, and the corresponding S56_37_7, with 2551 adaptable parameters. Note the good performance of the ASNN over its MLP counterpart.

In order to evaluate the generalization performance of the ASNN's, a simple test was also carried out. The test evaluates the $\langle \mathrm{MSE} \rangle$ and the percentage of correct character classification (hit-rate) for different realizations and percentage of salt and pepper noise. In particular, the networks trained with 2% of flipped pixels are tested with 3% and 5% of salt and pepper noise.

Each experiment was repeated three times using different weights initialization in order to obtain some statistical evidence. The results are reported in terms of both standard deviation (σ) and mean of the MSE_{dB} value ($\langle \mathrm{MSE} \rangle$ [dB]). Table I reports the result of this test.

B. Data Processing Experiments: Mackey–Glass Time Series Prediction

The first data processing experiment consists in a classical problem of prediction of a time-series, generated by the Mackey–Glass (MG) equation

$$\dot{x}(t) = \beta x(t) + \frac{\alpha x(t-\tau)}{1 + x(t-\tau)^{10}} \tag{25}$$

where values of the parameters are $\alpha = 0.2$, $\beta = -0.1$, and $\tau = 17$. The time-series $x(t)$ which originates from these parameters is quasiperiodic and chaotic, characterized by a 2.1 fractal dimension attractor [22].

Four time-series points $[x(t-18), x(t-12), x(t-6), x(t)]$ are considered as NN's input, while the one-dimension network desired output is the sample $[x(t+6)]$.

The learning phase consists of 200 epochs, where each epoch is composed of 4000 samples. The learning rate is $\mu = 0.002$ for each network considered in the experiments, equal for both weights and activation function control points. The runs compare standard MLP and ASNN networks, with the same number of adaptable parameters.

In the first trial five ASNN's and five standard MLP's are trained, the former with three hidden neurons and the latter with five hidden neurons. The networks have four inputs and a simple linear-combiner as output and are denoted as S4_3_1 and N4_5_1, respectively, both with 30 adaptable parameters.

For all the ASNN's the number of control points N is fixed $(N = 12)$ and $\Delta x = 1.2$. The initial activation functions for the ASNN's are standard sigmoids between zero and two. The activation functions of the MLP's are unipolar sigmoids between zero and two [$c_1 = 2$, $c_2 = 1$ and $c_3 = 0$ in (12)].

Fig. 6 shows a comparison among the learning curves of the two network families. Fig. 7 shows the learning curves for the same experiment using two different sets of networks: S4_6_1 and N4_10_1 both with 60 adaptable parameters. Again, note the performance of the proposed neural networks. Fig. 8 shows typical plots of the spline activation functions at the end of the learning phase.

In order to evaluate the generalization performance of the two network families, a simple prediction test was carried out. After the first prediction of the future time-series sample at time $(t+6)$ denoted as $\tilde{x}(t+6)$ (estimated value), we used this sample to feed the network's input delay line, in order to recursively predict the future samples following the scheme:

$$[x(t-18), x(t-12), x(t-6), x(t)] \rightarrow \tilde{x}(t+6)$$
$$[x(t-12), x(t-6), x(t), \tilde{x}(t+6)] \rightarrow \tilde{x}(t+12)$$
$$\cdots$$
$$[\tilde{x}(k-18), \tilde{x}(k-12), \tilde{x}(k+6), \tilde{x}(k)] \rightarrow \tilde{x}(k+12)$$
$$\cdots$$

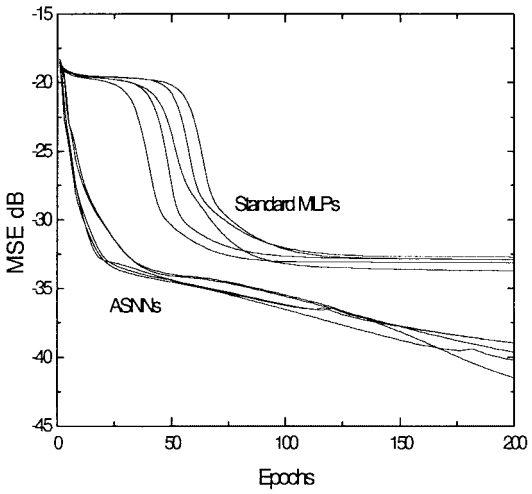


Fig. 6. Learning curve comparison between ASNN's (S4_3_1) and standard MLP (N4_5_1). Both networks with 30 adaptable parameters.

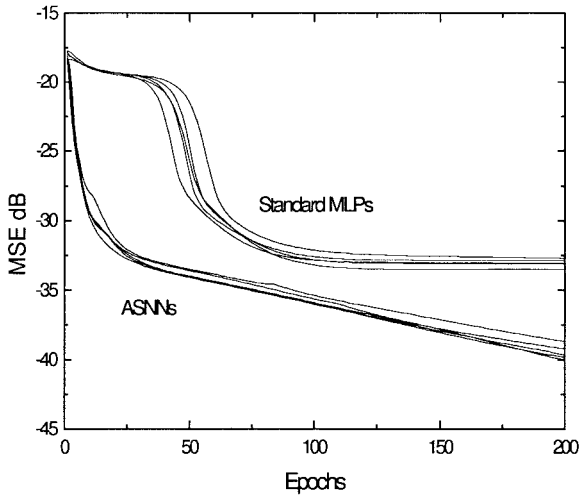


Fig. 7. Learning curve comparison between ASNN's (S4_6_1) and standard MLP (N4_10_1). Both networks with 60 adaptable parameters.

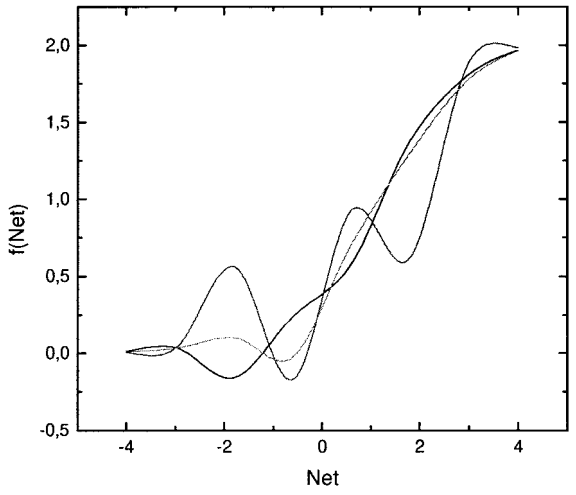
For that purpose we iteratively predicted $\tilde{x}(t+6)$, $\tilde{x}(t+12), \cdots$ until we reached $\tilde{x}(t+90)$ after 15 of such iterations. Table II reports $\langle \mathrm{MSE} \rangle$ error averaged over five previously trained networks, for the prediction of the future time-series samples at time $t = 30$, 60, and 90.

C. Data Processing Experiment: System Identification

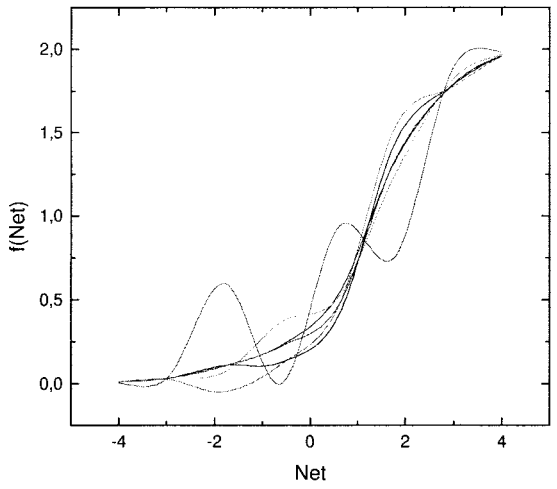
In the system identification test, each spline activation function has 28 control points initially sampled from a standard sigmoid [$c_1 = 2$, $c_2 = 1$, and $c_3 = 1$ in (12)], with $\Delta x = 0.5$.

The first set of experiments consists in identifying the nonlinear system with memory presented by Narendra in [23], described by the following state-space relationship:

$$x_1(k+1) = \frac{x_1(k) + 2x_2(k)}{1 + x_2^2(k)} + u(k)$$
$$x_2(k+1) = \frac{x_1(k)x_2(k)}{1 + x_2^2(k)} + u(k)$$
$$y(k) = (x_1(k) + x_2(k))/8 \tag{26}$$



(a)



(b)

Fig. 8. Typical activation function shapes after the learning process of the MG time series prediction. The SG-neurons parameters are: LUT length $N = 12$, $\Delta x = 1.2$, $\mu_w = \mu_q = 0.02$. (a) Network S4_3_1; (b) network S4_6_1.

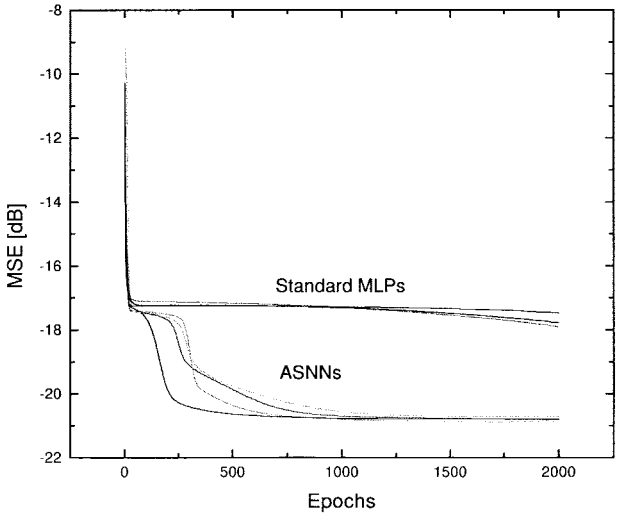
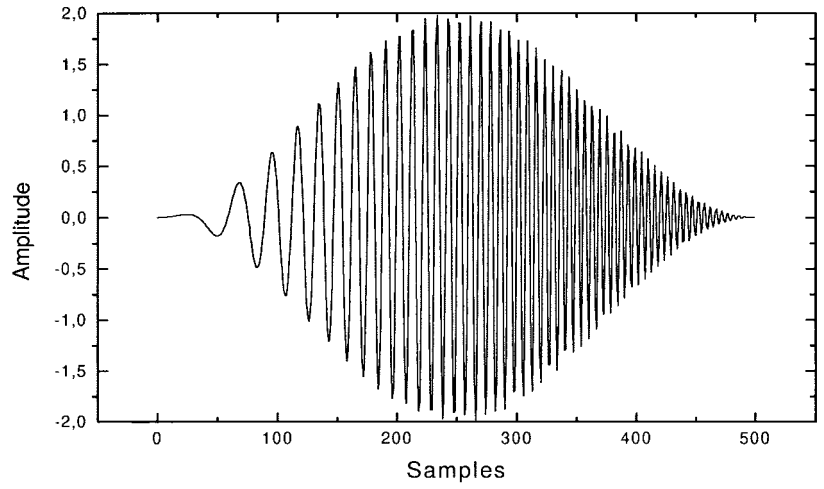


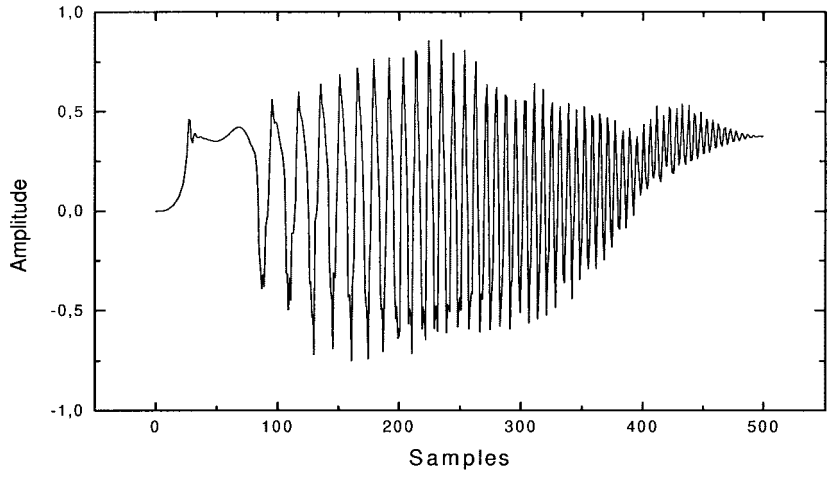
Fig. 9. MSE of two families of five networks with (about) the same numbers of adaptable parameters. ASNN's (S5_5_1) and MLP (N5_9_1) with 55 and 63 adaptable parameters, respectively.

TABLE II
MACKEY-GLASS TIME SERIES PREDICTION GENERALIZATION PERFORMANCE. THE TERM $\langle \mathrm{MSE} \rangle$ REPRESENTS THE AVERAGE MEAN SQUARE ERROR. THE NETWORKS SIZE ARE CHOOSE IN ORDER TO MAINTAIN EQUAL NUMBER OF ADAPTABLE PARAMETERS

Delay	Sigmoid MLP		ASNN	
-	N4_5_1	N4_10_1	S4_3_1	S4_6_1
-	<MSE> [dB] (σ)	<MSE> [dB] (σ)	<MSE> [dB] (σ)	<MSE> [dB] (σ)
$\Delta t = 30$	-36.93 (2.331)	-34.95 (3.954)	-39.65 (6.544)	-39.65 (0.501)
$\Delta t = 60$	-23.04 (0.839)	-21.71 (1.416)	-48.17 (6.309)	-48.17 (7.198)
$\Delta t = 90$	25.16 (1.875)	-24.09 (0.857)	-27.35 (4.309)	-27.35 (0.859)



(a)



(b)

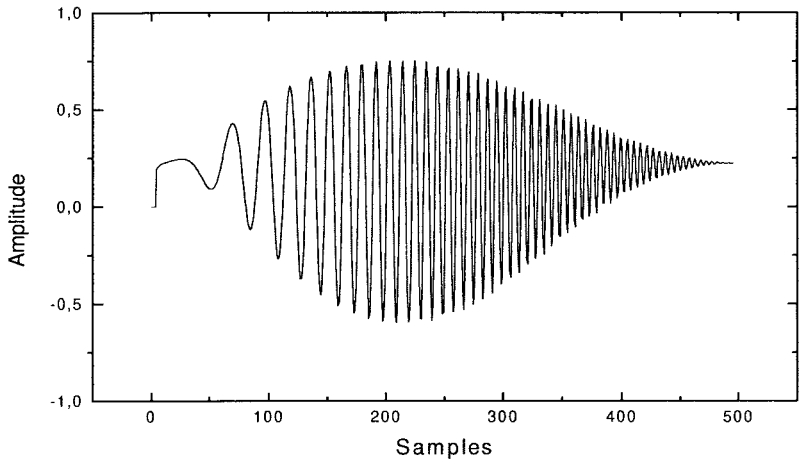
Fig. 10. Hanning windowed sweep signal test. (a) Sweep signal. (b) system output.

where $u(k)$ and $y(k)$ are the input and output signals at time k, respectively.

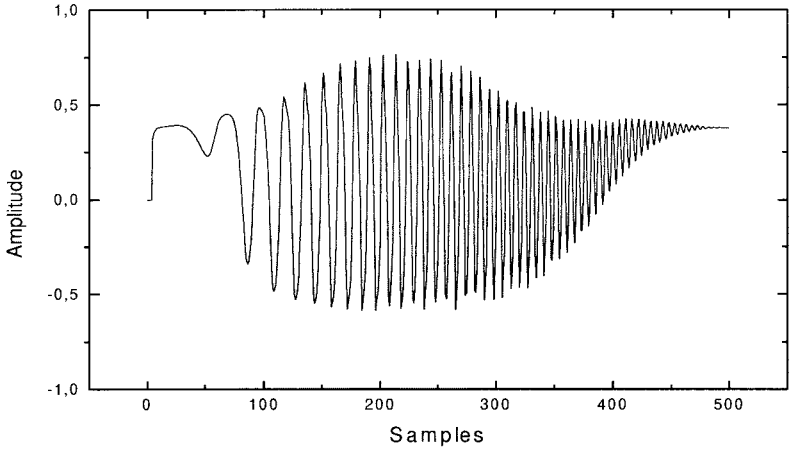
To provide the networks with the external memory necessary to identify the system, buffered ASNN's with five inputs are used: three network inputs are past system inputs, while the other two network inputs are the past system outputs. The activation function of the output neuron is linear.

A sequence of 1000 points of Gaussian random noise with zero mean and unit variance is generated as $u(k)$. This sequence and the corresponding output sequence $y(k)$ constitute the training set for the various neural networks. Two sets of five networks are used, denoted as N5_x_1 (MLP) and S5_x_1 (ASNN) where again x is the number of the hidden neurons $(x = 3, 5, 7, 9)$. The learning rate is $\mu_w = 0.008$ for all the networks. Moreover, for all the ASNN's, the activation function adaptation parameter is $\mu_q = 0.008$. The learning phase consists in 2000 epochs, each epoch being composed of 1000 samples.

Fig. 9 shows the MSE for two sets of networks with about the same numbers of adaptable parameters: S5_5_1 and N5_9_1 with 55 and 63 parameters, respectively. Moreover, in order to test the generalization capabilities, a forward phase is performed using a difficult input signal: a Hanning windowed sweep signal composed of 500 samples (see Fig. 10). The



(c)



(d)

Fig. 10. (*Continued.*) Hanning windowed sweep signal test. (c) N5_9_1 output signal. (d) S5_5_1 output signal.

TABLE III
PERFORMANCE COMPARISONS FOR THE NARENDRA SYSTEM IDENTIFICATION TEST

Number of hidden neurons	Sigmoid MLP		ASNN	
-	Training	Test	Training	Test
-	<MSE>[dB] (σ)	<MSE>[dB] (σ)	<MSE> [dB] (σ)	<MSE>[dB] (σ)
3	-19.72 (0.382)	-15.36 (0.353)	-20.69 (0.232)	-16.10 (0.234)
5	-19.80 (0.384)	-14.89 (0.302)	-20.80 (0.220)	-16.19 (0.228)
7	-20.32 (0.372)	-14.73 (0.401)	-21.30 (0.223)	-17.51 (0.213)
9	-19.43 (0.361)	-14.68 (0.371)	-21.36 (0.198)	-17.94 (0.204)

MSE obtained with the test set is then averaged on the five networks. The results are reported in Table III.

V. CONCLUSION

In this paper we have introduced a new neural network model based on an adaptive spline activation function, and its gradient-based learning algorithm. The main properties of this new architecture are: 1) universal approximation characteristics; 2) easy software and/or hardware implementation; 3) low-cost adaptation with constant computational complexity (only four control points for each neuron); 4) improvement on both the network structural complexity and the performance in terms of generalization capabilities.

Moreover, due to the locality of the chosen spline interpolation scheme, during the learning phase only four control points are adapted. As a consequence, the parameters modified at each iteration are only a subset of all the free parameters of the network.

It follows that new input data does not modify the whole shape of the neuron's output, but only a well-defined portion. This local character of the proposed adaptation scheme can be viewed as an application of the principle of minimal disturbance.

The experimental results on pattern recognition and data processing problems show that this model is very effective in terms of learning capabilities, while retaining a low level of complexity.

```
/* FORWARD COMPUTATION */
float C[4];                          /* CR polynomials storage */
float u, u2, u3;                      /* u, u^2, u^3 */
float tmp1,tmp2;                     /* temporary result */
u2 = u * u;
u3 = u2 * u;
tmp1 = u2 - u;
tmp2 = u3 - u2;
C [3] = ldexp(tmp2, -1);              /* (u3 - u2)/2 */
C [0] = ldexp (tmp1, -1) - C [3];
C [2] = C [0] + u - tmp2;
C [1] = 1 - C [0] - C [2] - C [3];

/* BACKWARD COMPUTATION */
/* u, u2, tmp1 already computed and stored during the forward phase */
float Cd[4];                          /* derivatives storage */
tmp3 = 0.5-u;
Cd [3] = tmp1 + ldexp(u2, -1);
Cd [1] = Cd [3] - u2 + ldexp(tmp1, 2);
Cd [2] = tmp3 - Cd [1];
Cd [0] = -Cd [1] - Cd [2] - Cd [3].
```

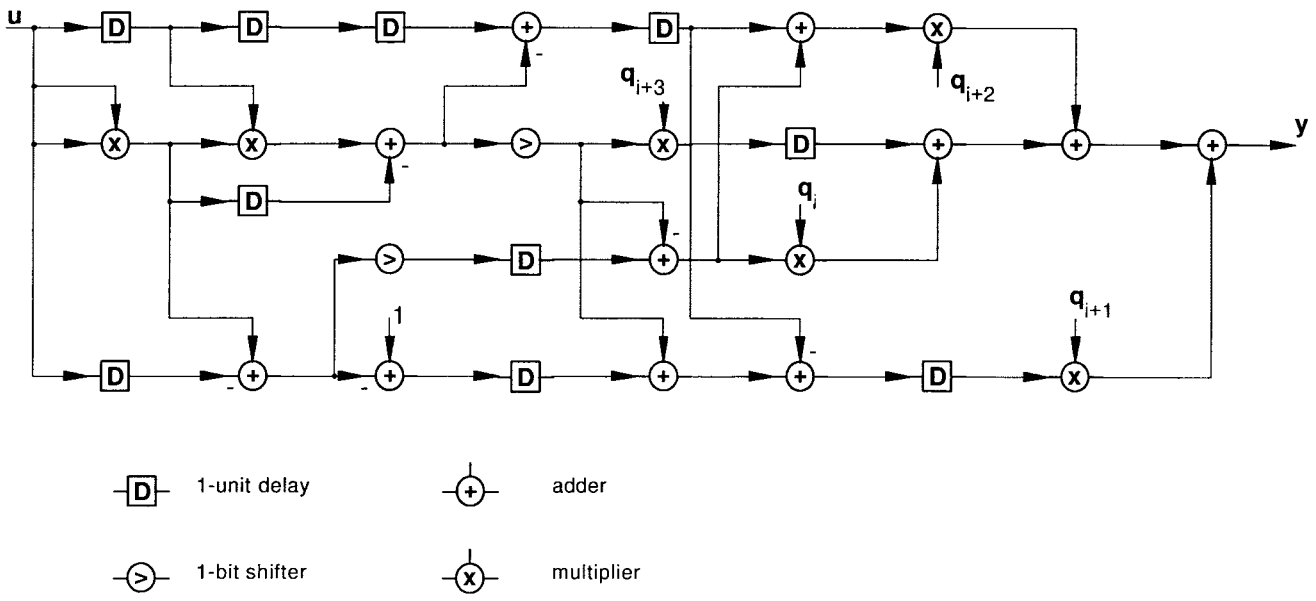


Fig. 11. Architecture of the circuit implementing the CR activation function computation.

APPENDIX
IMPLEMENTATION NOTES

Given a CR spline-based activation function, the most important optimization was achieved by using equally spaced control point abscissas, leading to a fast computation of the span index i and of the offset variable u (see GS1 block). However, this block can be further optimized, choosing the sampling step as a power of two ($\Delta = {}^{b}$with b signed integer) and the number of control points odd (N even). In fact, these constraints transform the two multiplications in the second of (13) into two shifts, namely

$$z_k^{(l)} = \left(s_k^{(l)} \ll b\right) + (N \ll 2) - 1 \tag{A1}$$

where $\ll$ is the shift operator.

The GS2 block can also be further optimized. Referring to (5), we see that the neuron output is calculated through a sum of weighted control point ordinates, the weights being the CR cubic polynomials (4). Unfortunately, this calculation step has to be executed four times for each neuron in the network during the forward phase, as well as four more times (the CR cubic derivatives) in the backward phase. However, noting that all the polynomial coefficients of the CR cubic spline are powers of two or sums of two powers of two, most of the computations can be done efficiently only with sums and shifts, and also through the reuse of past calculations. The following two C code fragments show how to efficiently implement the forward and backward computations, respectively (the fact that the sum of the basis functions is identically equal to one was

exploited in the code); as shown at the top of the previous page. A comparison of the proposed algorithm with the well-known Horner's rule, in terms of performed operations per single GS2 block in both forward and backward calculations, shows a complexity reduction of about 30% in terms of number of operations, with a tenfold reduction of the multiply operations (from 20 to 2), substituted by simpler sums and shifts.

The reduction of multiplications can also be considered in hardware. Fig. 11 reports the signal flow-graph of a parallel implementation of the GS2 output computation. The operations are ordered in time slices (from left to right), which are represented as columns of cells. Note the good performance of the structure both in terms of parallelism and time slices. Although a rather large amount of chip area is required for a single GS2 activation function, the reduction in structural complexity that ASNN's can attain must be taken into account: in fact, the complex linear combiner stages are now greatly reduced, both in length and in number, due to the reduction of the number of neurons per layer.

REFERENCES

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas imminent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.

[2] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing (PDP): Exploration in the Microstructure of Cognition*, vol. 1. Cambridge, MA: MIT Press, 1986.

[3] Widrow and M. Lehr, "30 years of adaptive neural networks: Perceptron, adaline and backpropagation," in *Proc. IEEE*, vol. 78, no. 9, Sept. 1990.

[4] C. Z. Tang and H. K. Kwan, "Convergence and generalization properties of multilayer feedforward neural networks," in *Proc. IEEE Int. Symp. Circuits Syst.*, San Diego, CA, 1992, pp. I-65–68.

[5] C. T. Chen and W. D. Chang, "A feedforward neural network with function shape autotuning," *Neural Networks*, vol. 9, no. 4, pp. 627–641, June 1996.

[6] G. Cybenko, "Approximation by superposition of a sigmoidal function," *Math. Contr. Signals Syst.*, no. 2, 1989.

[7] T. Chen and H. Chen, "Approximation of continuous functionals by neural networks with application to dynamic systems," *IEEE Trans. Neural Networks*, vol. 4, pp. 910–918, Nov. 1993.

[8] T. Chen, H. Chen, and R. Liu, "Approximation capability in $C(\mathbf{R}^n)$ by multilayer feedforward networks and related problems," *IEEE Trans. Neural Networks*, vol. 6, pp. 25–30, Jan. 1995.

[9] K. Hornik, M. Stinchombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[10] ———, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, pp. 551–560, 1990.

[11] M. Stinchcombe and H. White, "Universal approximation using feedforward networks with nonsigmoid hidden layer activation functions," in *Proc. IJCNN*, Washington, D.C., 1989, pp. I 613–617.

[12] ———, "Approximating and learning unknown mappings using multilayer feedforward networks with bounded weights," in *Proc. IJCNN*, San Diego, CA, 1990, pp. III 7–16.

[13] S. Amari and N. Murata, "Statistical theory of learning curves under entropic loss criterion," *Neural Comput.*, vol. 5, pp. 140–153, 1993.

[14] S. Amari, "A universal theorem on learning curves," *Neural Networks*, vol. 6, no. 2, pp. 161–166, 1993.

[15] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *IEEE Trans. Neural Networks*, vol. 3, pp. 101–120, Nov. 1992.

[16] Piazza, A. Uncini, and M. Zenobi, "Artificial neural networks with adaptive polynomial activation function," in *Proc. IJCNN*, Beijing, China, Nov. 1992, pp. II-343–349.

[17] ———, "Neural networks with digital LUT activation function," in *Proc. IJCNN*, Nagoya, Japan, 1993, pp. II-1401–1404.

[18] I. J. Schoenberg, "Contributions to the problem of approximation of equidistant data by analytic functions," *Quart. Appl. Math.*, vol. 4, pp. 45–99, 1946.

[19] R. E. Barnhill and R. F. Riesenfeld, Eds., *Computer Aided Geometric Design*. New York: Academic, 1974.

[20] E. Catmull and R. Rom, "A class of local interpolating splines," in *Computer-Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld, Eds. New York: Academic, 1974, pp. 317–326.

[21] F. Piazza, S. Smerilli, A. Uncini, M. Griffo, and R. Zunino, "Fast spline neural networks for image compression," in *Proc. IX Wrkshp. Parallel Architectures and Neural Networks*, V. S. Mare (SA), Ed. Berlin, Germany: Springer-Verlag, May 1996.

[22] M. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, p. 287, 1977.

[23] K. S. Narendra, "Adaptive control of dynamical systems using neural networks," *Handbook of Intelligent Control*. New York: Van Nostrand Reinhold, 1992, pp. 141–183.

[24] J.-N. Hwang, S.-R. Lay, M. Maechler, R. D. Martin, and J. Schimert, "Regression modeling in backpropagation and projection pursuit learning," *IEEE Trans. Neural Networks*, vol. 5, pp. 342–353, 1994.

[25] F. Filippetti, A. Uncini, F. Piazza, P. Campolucci, C. Tassoni, and G. Franceschini, "Neural-network architectures for fault diagnosis and parameter recognition in induction machines," in *Proc. IEEE Melecon'96*, Bari, Italy, May 1996.

Stefano Guarnieri was born in Ancona, Italy, in March 1969. He received the laurea degree with honors in electronic engineering from the University of Ancona, Italy, in 1994.

He is currently working in The MacGregor Group Inc., a Boston, MA-based trading systems software company. His current research interests include the applicability of the neural networks to trading systems modeling.

Francesco Piazza (M'87) was born in Jesi, Italy, in February 1957. He received the laurea degree with honors in electronic engineering from the University of Ancona, Italy, in 1981.

From 1981 to 1983 he worked on CCD-image processing at the Physics Department of the University of Ancona. In 1983 he worked at the Olivetti OSAI software development center. In 1984 he was a Consultant in the field of microprocessor-based systems and radar displays. In 1985 he joined the Department of Electronics and Automatics of the University of Ancona, first as Researcher in Electrical Engineering and then as Associate Professor. His current research interests include circuit theory and digital signal processing and include adaptive DSP algorithms and circuits, neural networks, and speech and audio processing.

Aurelio Uncini (M'88) received the laurea degree in electronic engineering from the University of Ancona, Italy, and the Ph.D. degree in electrical engineering from University of Bologna, Italy, in 1983 and 1993 respectively.

From 1984 to 1986 he was with the "Ugo Bordoni" Foundation, Rome, Italy, engaged in research on digital processing of speech signal. From 1986 to 1987 he was at Italian Ministry of Communication. From 1994 to 1998 he was a Researcher at the Electronics and Automatics Department at the University of Ancona. Currently he is Associate Professor of Electrical Engineering at the INFOCOM Department, University of Rome, "La Sapienza." His research interests are in digital signal processing, neural networks, circuit theory. His present research interests include adaptive filters, audio and speech processing, optimization algorithms for circuits design, and neural networks for signal processing.