Reprinted from

IJCNN-2000
Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Network,
Como, Italy, 24-27 July, 2000

Artificial Neural Networks with Adaptive Multidimensional Spline Activation Functions

Mirko Solazzi(*), Aurelio Uncini(**)

(*) Dipartimento di Elettronica e Automatica - University of Ancona
Via Brecce Bianche, 60131 Ancona-Italy.
Fax:+39 (071) 2204464 - email: mrksol@eealab.unian.it

(**) Dipartimento INFOCOM - University of Rome "La Sapienza"
Via Eudossiana 18, 00184 Rome - Italy.
Fax +39 (06) 4873300 email: aurel@infocom.uniroma1.it

Artificial Neural Networks with Adaptive Multidimensional Spline Activation Functions

Mirko Solazzi(*), Aurelio Uncini(**)

(*) Dipartimento di Elettronica e Automatica - University of Ancona
Via Brecce Bianche, 60131 Ancona-Italy.
Fax:+39 (071) 2204464 - email: mrksol@eealab.unian.it - Internet: http://nnsp.eealab.unian.it/

(**) Dipartimento INFOCOM - University of Rome "La Sapienza"
Via Eudossiana 18, 00184 Rome - Italy.
Fax +39 (06) 4873300 email: aurel@infocom.uniroma1.it

ABSTRACT

This work concerns a new kind of neural structure that involves a multidimensional adaptive activation function. The proposed architecture, based on multi-dimensional cubic spline, allows to collect information from the previous network layer in aggregate form. In other words the number of network connections (structural complexity) can be very low respect to the problem complexity. This fact, as experimentally demonstrated in the paper, improve the network generalization capabilities and speed up the convergence of the learning process. A specific learning algorithm is derived and experimental results, demonstrate the effectiveness of the proposed architecture.

I. INTRODUCTION

In traditional multilayer feed-forward neural networks each neuron computes the weighted sum of its inputs and applies to this sum a non-liner function called activation functions. Such functions are univariate, in other words, they represent a relation between a single input, the weighted sum, and a single output, the neuron response. Structures with multiple layers guarantee approximation capability of complex functions of several inputs.
Recently a new interest in adaptive activation functions has arisen. 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 [1]. A different approach is based on the use of polynomial functions [2], which allows to reduce the size of the network and, in particular, the connection complexity; in fact, the digital implementation of the activation function through a LUT (look-up-table) keeps the overall complexity under control. This solution implies some drawbacks, principally with the adaptation of the coefficients in the learning phase, due to spurious minima and maxima. In addition, a polynomial function is a non-bounded function and the resulting approximation is generally poorly smooth. Later adaptive spline activation function was introduced [3]. With this approach each neuron is characterized by a different activation function whose shape can be modified through some control points. Further works [4][5] demonstrated that such neuron architecture can improve approximation and generalization abilities of the entire network.
The main idea of this work is to design a new kind of neural structure that involves more complex neurons. Specifically we introduce multivariate activation functions, in other terms the neuron response is the result of combination of several values, obtained as weighted sum of the layer inputs. Bivariate activation functions are studied in detail. Referring to Vitushkin's theorem [6], the new network structure can approximate any functions of two variables and of course it can better represent functions of several variables. The major advantage in this approach in given by the possibility to implement these activation functions using cubic spline interpolation of control points.

II. REGULARIZATION THEORY AND MULTIDIMENSIONAL APPROXIMATION

The problem of learning a smooth mapping from examples is ill-posed in the sense that information in the data is generally not sufficient to reconstruct uniquely the mapping in regions where data are not available. Moreover, the present of noise in examples can lead to discover spurious structure in the data. Regularization techniques impose smoothness constraints on the approximating set of function *f*, excluding high-frequency components. This allows increasing the generalization capability in approximation type problems.
Typically, regularization theory follows the classical technique introduced by Tikhonov [7][8]. This technique consists in introducing an additional smoothness functional $\phi[f]$, called *stabilizer* into the cost functional $E[f]$ to be minimized:

$$E[f]=\sum_{n=1}^{N_s}\left(t_n - f(\mathbf{x}_n)\right)^2 + \lambda\phi[f] \qquad (1)$$

In the above expression, λ is a positive number usually named the *regularization parameter* and $(\mathbf{x}_n, t_n) \in R^L \times R$ with $1 \le n \le N_e$ are N_e pairs of training data obtained by random sampling a function *f*, belonging to some function space defined on R^L. The first term enforces the data fitting while the second guarantees an appropriate degree of smoothness. The high frequency content, measured by first high-pass filtering the function power spectrum, can be used as stabilizer:

$$\phi[f] = \int_{R^L} \frac{|F(\omega)|^2}{G(\omega)} d\omega \tag{2}$$

where $1/G(\omega)$ is the transfer function of the high-pass filter and $F(\omega)$ the Fourier transform of function *f*. Minimization of the cost functional $E[f]$ leads to a well-known solution given by [9]:

$$f(\mathbf{x}) = \sum_{n=1}^{N_e} c_n g(\mathbf{x} - \mathbf{x}_n) \tag{3}$$

where *g* is the chosen basis function. The coefficients c_n depend on the data, and can be determined by solving the $N_e \times N_e$ linear system $(\mathbf{G} + \lambda \mathbf{I})\mathbf{c} = \mathbf{t}$, where $(\mathbf{G})_{mn} = g(\mathbf{x}_m - \mathbf{x}_n)$ is the Green's matrix, $(\mathbf{c})_n = c_n$ is the column vector of the coefficients and $(\mathbf{t})_n = t_n$ is the column vector of the targets. In real application, where the number of examples N_e is usually large, the problem related to the inversion of the matrix $(\mathbf{G} + \lambda \mathbf{I})$ results very difficult to solve. To avoid massive calculations we should think a network implementation of equation (3), using adaptive techniques to find the coefficients.
As regard the choice of basis function, it depends on the class of stabilizers adopted. It is possible to derive more general approximation schemes in the same framework considering a tensor product of basis functions or an additive scheme. In particular, we should consider the function *f* as superposition of one-dimensional functions, one for each component of the input vector **x**, so that the effects of the different input variables can be examined separately [9]. We want here extend this solution considering a decomposition the function *f* as sum of $N = \lceil L/M \rceil$ *M*-dimensional functions, each in charge of *M* components of the vector $\mathbf{x}$ ($\lceil . \rceil$ is the ceiling operator). We mean an approximation of the form:

$$f(\mathbf{x}) = \sum_{j=0}^{N-1} f_j(x_{M \cdot j+1}, x_{M \cdot j+2}, ..., x_{M \cdot j+M}) \tag{4}$$

Since reminder of division L/M can be non zero, some inputs for the last function are arbitrarily fixed.
The simplest way to obtain this kind of approximation scheme is to choose a stabilizer such that it results:

$$g(\mathbf{x}) = \sum_{j=0}^{N-1} \mu_j g(x_{M \cdot j+1}, x_{M \cdot j+2}, ..., x_{M \cdot j+M}) \tag{5}$$

where μ_j are fixed parameters and $g(x_1, x_2, ..., x_M)$ are an M-dimensional basis function. Substituting (5) in (3) we get to the final expression of the approximation function:

$$f(\mathbf{x}) = \sum_{n=1}^{N_e} c_n \sum_{j=0}^{N-1} \mu_j g((x_{M \cdot j+1} - x_{n;M \cdot j+1}), ..., (x_{M \cdot j+M} - x_{n;M \cdot j+M})) \tag{6}$$

The symbol $x_{n;M \cdot j+k}$ represents the $Mj+k$ component ($1 \le k \le M$) of the *n*-th sample data.
As frequently occurs in practical applications, some of the input variables can be more relevant respect to others. It can therefore be appropriate to consider a change in the system of coordinates for the input space introducing a linear transformation. Calling $\mathbf{w}_{k,j}$, $1 \le k \le M$ and $0 \le j \le N-1$ the vectors which determine the axis of the new system for *k*-th input variable of *j*-th function and $\alpha_{n;k,j}$ the new centers in such a system, we obtain:

$$f(\mathbf{x}) = \sum_{j=0}^{N-1} \mu_j \sum_{n=1}^{N_e} c_n g(\mathbf{w}_{1,j}\mathbf{x} - \alpha_{n;1,j}, ..., \mathbf{w}_{M,j}\mathbf{x} - \alpha_{n;M,j}) \tag{7}$$

Defining the basis function for the new system of coordinates as:

$$\psi_j(\mathbf{w}_{1,j}\mathbf{x}, .., \mathbf{w}_{M,j}\mathbf{x}) = \sum_{n=1}^{N_e} c_n g(\mathbf{w}_{1,j}\mathbf{x} - \alpha_{n;1,j}, ..., \mathbf{w}_{M,j}\mathbf{x} - \alpha_{n;M,j}) \tag{8}$$

we can rewrite the expression (7) as:

$$f(\mathbf{x}) = \sum_{j=0}^{N-1} \mu_j \psi_j(\mathbf{w}_{1,j}\mathbf{x}, ..., \mathbf{w}_{M,j}\mathbf{x}) \tag{9}$$

III. NEURAL NETWORKS WITH MULTI-DIMENSIONAL NEURONS

The approximation scheme derived from the regularization theory can be revised into the neural network contest. The aim of this process is to exploit the neural network optimization techniques to estimate the parameters μ and the basis functions Ψ. A multi-layer perceptron is a feed-forward neural network, consisting of a given number of units called neurons, which are connected by weighted links. These units are arranged in multiple layers, namely an input layer, one or more hidden layers and an output layer; in total *S* layers. The overall network function that maps the input vector onto the output vector is established by the network connection weights. Each neuron *j* in the layer *l* of the network is a simple processing unit whose output is computed by passing the net input through a non-linear activation function $\Phi^{(l)}(x)$:

$$y_j^{(l)} = \Phi^{(l)}\left(net_j^{(l)}\right) = \Phi^{(l)}\left(\sum_{i=0}^{N^{(l-1)}} w_{ji}^{(l)}\, y_i^{(l-1)}\right) \tag{10}$$

A generalization of this structure in now introduced, using more general multi-dimensional activation functions. Let $\Phi_j^{(l)}(x_1, x_2, ..., x_M)$ generic functions of *M* variables, we define multiple net inputs (see Figure 1):

$$net_{k,j}^{(l)} = \sum_{i=0}^{N^{(l-1)}} w_{k,ji}^{(l)} y_i^{(l-1)} = \mathbf{w}_{k,j}^{(l)} \mathbf{y}^{(l-1)} \tag{11}$$

such that the activation of neuron *j* in the layer *l* is computed as:

$$y_j^{(l)} = \Phi_j^{(l)}\left(net_{1,j}^{(l)}, .., net_{M(l),j}^{(l)}\right) = \Phi_j^{(l)}\left(\sum_{i=0}^{N^{(l-1)}} w_{1,ji}^{(l)} y_i^{(l-1)}, ..., \sum_{i=0}^{N^{(l-1)}} w_{M(l),ji}^{(l)} y_i^{(l-1)}\right) = \Phi_j^{(l)}\left(\mathbf{w}_{1,j}^{(l)}\mathbf{y}^{(l-1)}, ..., \mathbf{w}_{M(l),j}^{(l)}\mathbf{y}^{(l-1)}\right) \tag{12}$$

If we now consider a two-layer structure with linear output activation functions, the network functions are:

$$f_i(\mathbf{x}) = \sum_{k=1}^{M(2)} \alpha_k \sum_{j=0}^{N^{(1)}} w_{k,ij}^{(2)} y_j^{(1)} = \sum_{j=0}^{N^{(1)}} \left(\sum_{k=1}^{M(2)} \alpha_k w_{k,ij}^{(2)}\right) \Phi_j^{(1)}\left(\mathbf{w}_{1,j}^{(1)}\mathbf{x}, ..., \mathbf{w}_{M(1),j}^{(1)}\mathbf{x}\right) = \sum_{j=0}^{N^{(1)}} \mu_j^{(i)} \Phi_j^{(1)}\left(\mathbf{w}_{1,j}^{(1)}\mathbf{x}, ..., \mathbf{w}_{M(1),j}^{(1)}\mathbf{x}\right) \tag{13}$$

Equation (13) has the same form than expression (9). Using this network representation, we can find the solution of any function approximation problem in the regularization framework without necessarily invert the matrix $(\mathbf{G} + \lambda\mathbf{I})$. It's sufficient to apply standard network learning techniques to find appropriate parameters of approximation scheme. In particular we can consider activation functions $\Phi_j^{(l)}(.)$ as basis functions $\psi_j(.)$.

IV. CUBIC SPLINE BASED ADAPTIVE ACTIVATION FUNCTION

The network implementation of equation (9) requires the definition of M-dimensional basis functions ψ_j. Our idea consists in realizing these functions as hyper-surface interpolation of some control points using higher order interpolants. In particular piecewise cubic splines are here employed in order to render the hyper-surface continuous in its first partial derivatives. The entire approximation is represented through the concatenation of local functions $h_{n;j}(u_1, u_2, .., u_M)$, each related to center $\alpha_{n;k,j}$,$(1 < k < M)$ and controlled by 4^M control points. For example, a 1D basis function $h_{n;j}(u_1)$ is specified over the interval $u_1^{(n_1)} \le u_1 \le u_1^{(n_1+1)}$ as the weighted average of the four points $Q_j^{/n_1-1)}, Q_j^{/n_1)}, Q_j^{/n_1+1)}, Q_j^{/n_1+2)}$ which are equally spaced along the u_1 axis. Without loss of generality, the origin can be translated to the point $u_1^{(n_1)}$ and the interval scaled such that u_1 varies over the range $0 \le u_1 \le 1$. In 2D, basis functions $h_{n;j}(u_1, u_2)$ are defined over the region $0 \le u_1, u_2 \le 1$, as the weighted average of the sixteen points $\left\{Q_j^{(n_1+s_1-1, n_2+s_2-1)}\right\}_{s_1=(0,1,2,3),\ s_2=(0,1,2,3)}$, which lie on a regular 2D grid in R^2. Indices n_k are related to the *k*-th coordinate of centers $\alpha_{n;k,j}$. The general expression is given by:

$$h_{n;j}(u_1, u_2, .., u_M) = \sum_{s_1=0}^{3} \sum_{s_2=0}^{3} ... \sum_{s_M=0}^{3} \rho_{s_1, s_2, ..., s_M; j}\; u_1^{s_1} u_2^{s_2} \cdot\cdot u_M^{s_M} \tag{14}$$

where $\rho_{s_1, s_2, ..., s_M; j}$ are coefficients depending on control points of the *j*-th basis function.
Synthetic formulation for 1D and 2D domain can be done by mean of matrix representation [10]:

$$h_{n;j}(u_1) = \mathbf{T}_1 \cdot \mathbf{M} \cdot \mathbf{Q}_{j[1]}^{(n)} \qquad h_{n;j}(u_1, u_2) = \mathbf{T}_2 \cdot \mathbf{M} \cdot \left(\mathbf{T}_1 \cdot \mathbf{M} \cdot \mathbf{Q}_{j[2]}^{(n)}\right)^T \tag{15}$$

where: $\mathbf{T}_k = \begin{bmatrix} u_k^3 & u_k^2 & u_k & 1 \end{bmatrix}$ with $0 \le u_k \le 1$, $\forall k : 1 \le k \le M$ and $\mathbf{Q}_{j[M]}^{(n)}$ is a *M*-dim structure that collect the local control points of the *j*-th basis function:

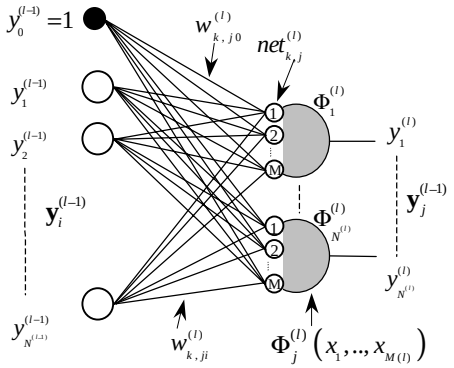

Figure 1. Architecture of a neural network with multi dimensional neurons.

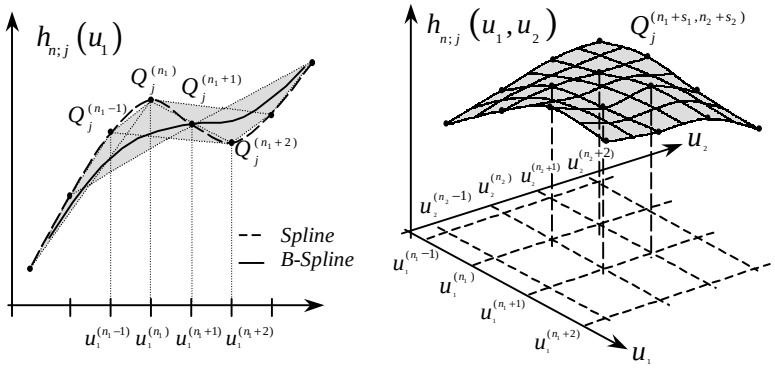

Figure 2. Examples of cubic spline interpolation of control points in 1D and 2D.

$$\mathbf{Q}_{j[1]}^{(n)} = \begin{bmatrix} Q_j^{(n_1-1)} \\ Q_j^{(n_1)} \\ Q_j^{(n_1+1)} \\ Q_j^{(n_1+2)} \end{bmatrix}; \qquad \mathbf{Q}_{j[2]}^{(n)} = \begin{bmatrix} Q_j^{(n_1-1,n_2-1)} & Q_j^{(n_1-1,n_2)} & Q_j^{(n_1-1,n_2+1)} & Q_j^{(n_1-1,n_2+2)} \\ Q_j^{(n_1,n_2-1)} & Q_j^{(n_1,n_2)} & Q_j^{(n_1,n_2+1)} & Q_j^{(n_1,n_2+2)} \\ Q_j^{(n_1+1,n_2-1)} & Q_j^{(n_1+1,n_2)} & Q_j^{(n_1+1,n_2+1)} & Q_j^{(n_1+1,n_2+2)} \\ Q_j^{(n_1+2,n_2-1)} & Q_j^{(n_1+2,n_2)} & Q_j^{(n_1+2,n_2+1)} & Q_j^{(n_1+2,n_2+2)} \end{bmatrix} \tag{16}$$

Figure 2 shows interpolation of control points in 1D and 2D. The matrix $\mathbf{M}$ determines the characteristic of the interpolant hyper-surface. The spline matrix impose the hyper-surface to pass through the control points; using a B-spline matrix the interpolant surface is constrained to lie within the *convex hull* related to the control points instead.

$$\mathbf{M} = \frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \text{ (Spline)}; \qquad \mathbf{M} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \text{ (B-Spline)} \tag{17}$$

Referring to the network structure illustrated in Figure 1, we derive an algorithm to compute the activation of each neuron *j* during presentation of the *n*-th example. Localization of net input vector is performed considering control points in each direction that are equally spaced and symmetrically disposed respect to the axis origin. Let Δx_k and N_k the fixed step between points and the number of points along the *k*-th dimension respectively, we have:

$$z_{k,j} = \frac{net_{k,j}}{\Delta x_k} + \frac{N_k - 2}{2}, \quad \text{with the constraint:} \qquad z_{k,j} = \begin{cases} 1 & \text{if } z_{k,j} < 1 \\ z_{k,j} & \text{if } 1 \le z_{k,j} \le N_k - 3 \\ N_k - 3 & \text{if } z_{k,j} > N_k - 3 \end{cases} \tag{18}$$

The constraint imposed is necessary to keep the net input within the active volume that encloses the control points. Next step is to separate indices $z_{k,j}$ into integer and fractional part using the floor operator $\lfloor . \rfloor$:

$$n_{k;j} = \lfloor z_{k,j} \rfloor; \qquad u_{k,j} = z_{k,j} - n_{k;j} \tag{19}$$

The integer part indicated as $n_{k;j}$ is used to address local control points of neuron *j*, while the fractional part $u_{k,j}$ is passed as normalized input to multidimensional cubic spline function $h_{n;j}$. After the training with N_e examples, activation function for the *j*-th neuron can be expressed as a combination of local spline basis functions:

$$\Phi_j^{(l)}\left(\mathbf{w}_{1,j}\mathbf{x},..,\mathbf{w}_{M,j}\mathbf{x}\right) = \Phi_j^{(l)}\left(net_{1,j},..,net_{M,j}\right) = \sum_{n=1}^{Ne} h_{n;j}\left(u_{1,j},u_{2,j},...,u_{M,j}\right) \tag{20}$$

V. LEARNING ALGORITHM

The backpropagation (BP) algorithm is a well-known method to tune the weights such that the network performs a specific mapping of the input vector onto a desired output vector. An extension of the classical BP algorithm for multi-dimensional spline networks is now derived. Being *p* the learning step, the weights in the network are updated along a search direction opposite of the cost function gradient:

$$w_{k,ji}^{(l)}[p+1] = w_{k,ji}^{(l)}[p] + \Delta w_{k,ji}^{(l)}[p] = w_{k,ji}^{(l)}[p] - \eta \frac{\partial E}{\partial w_{k,ji}^{(l)}}[p] = w_{k,ji}^{(l)}[p] + \eta \delta_{k,j}^{(l)} y_i^{(l-1)}[p] \tag{21}$$

$$\delta_{k,j}^{(l)} = e_j^{(l)} \cdot \left. \frac{\partial \Phi_j^{(l)}\left(x_1,..,x_{M(l)}\right)}{\partial x_k} \right|_{\left(x_1 = net_{1,j}^{(l)}, x_2 = net_{2,j}^{(l)},..., x_{M(l)} = net_{M(l),j}^{(l)}\right)} \tag{22}$$

$$e_j^{(l)} = \begin{cases} t_j - y_j^{(S)} & l = S \\ \sum_{k=1}^{M(l)} \left(\sum_{n=1}^{N^{(l+1)}} \delta_{k,n}^{(l+1)} w_{k,nj}^{(l+1)} \right) & l = S-1,..,1 \end{cases} \tag{23}$$

with $1 \le j \le N^{(l)}$, $0 \le i \le N^{(l-1)}$ and $1 \le k \le M(l)$. In particular, for multidimensional spline neurons have:

$$\delta_{k,j}^{(l)} = e_j^{(l)} \cdot \left. \frac{\partial \psi_j^{(l)}\left(x_1,..,x_{M(l)}\right)}{\partial x_k} \right|_{\left(x_1 = net_{1,j}^{(l)},...\right)} = e_j^{(l)} \sum_{n=1}^{p} \left(\left. \frac{\partial h_{n;j}^{(l)}\left(u_1,..,u_{M(l)}\right)}{\partial u_k} \right|_{\left(u_1 = u_{1,i}^{(l)},...\right)} \right) \frac{1}{\Delta x_k^{(l)}} \tag{24}$$

where $e_j^{(l)}$ is defined as in expression (23).

$$\frac{\partial h_{n;j}^{(l)}\left(u_1,u_2\right)}{\partial u_1} = \left(\mathbf{T}_2 \cdot \mathbf{M} \cdot \left(\dot{\mathbf{T}}_1 \cdot \mathbf{M} \cdot \mathbf{Q}_{j[2]}^{(n)} \right)^T \right)_j^{(l)} \quad ; \qquad \frac{\partial h_{n;j}^{(l)}\left(u_1,u_2\right)}{\partial u_2} = \left(\dot{\mathbf{T}}_2 \cdot \mathbf{M} \cdot \left(\mathbf{T}_1 \cdot \mathbf{M} \cdot \mathbf{Q}_{j[2]}^{(n)} \right)^T \right)_j^{(l)} \tag{25}$$

where $\dot{\mathbf{T}}_k = \begin{bmatrix} 3u_k^2 & 2u_k & 1 & 0 \end{bmatrix}$.

Control points of spline activation functions $h_{n;j}^{(l)}$ are also updated at each step using the adaptation rule:

$$\left(Q_j^{(n_1+s_1-1,n_2+s_2-1,...)} \right)^{(l)}[p+1] = \left(Q_j^{(n_1+s_1-1,n_2+s_2-1,...)} \right)^{(l)}[p] + \left(\eta_Q e_j^{(l)} \sum_{n=1}^{p} \left(\frac{\partial h_{n;j}^{(l)}\left(u_1,..,u_{M(l)}\right)}{\partial Q_j^{(n_1+s_1-1,n_2+s_2-1,...)}} \right) y_i^{(l-1)} \right)^{(l)}[p] \tag{26}$$

with $s_1 = (0,1,2,3)$, $s_2 = (0,1,2,3),....$. The number of points that we have to adjust is $4^{M(l)}$.

VI. EXPERIMENTAL RESULTS - CONCLUSIONS

Several experiments have been performed in order to compare the classical sigmoid and the mono-dimensional spline based activation function with the proposed multidimensional neuron. Two principal aspects were taken into account: the generalization capability and the control of the number of free parameters. While in a sigmoidal network all the parameters are updated at every step of the learning algorithm, using spline neurons only a subset of them results modified. As adaptation is made on a fixed number of the control points for each input sample, the number of control points can be initially chosen arbitrarily. The results of simulations on two-dimensional functions are reported in Table 1 and Table 2. The first function is:

$$f(x,y) = \sin(2\pi x) + 4(y-0.5)^2 \tag{27}$$

while the second is the Gabor function defined as:

$$f(x,y) = e^{-(x^2+y^2)} \cos\left[0.75\pi(x+y)\right] \tag{28}$$

First columns indicate the different types of tested networks: *N_x* stands for a classical MLP with *x* hidden units, while *S1_x* and *S2_x* are respectively one-dimensional and two-dimensional spline networks with *x* hidden neurons. Second columns contain the number of parameters updated at every learning step *(N_l)*. The tests were performed stopping the training when -20dB, -22dB or -24dB error levels were reached during the learning phase. The learning set was composed by 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 criteria for the computation of generalization error mean, expressed in dB, and variance (<g.e.> and $\sigma_{g.e.}^2$) is based on the initialization of twenty networks: if at least half of this group reached the threshold for training stop, <g.e.> and $\sigma_{g.e.}^2$ was calculated averaging on the best 10 networks over an equally spaced grid of 441 points in [-1,+1]× [-1,+1], otherwise no values were reported in the corresponding columns. It was also considered the situation after 1000 learning cycles.

In order to evaluate the generalization capability, another experiment was been carried out with the classical "two-spirals" problem [11]. It is an extremely hard two-class problem for plain MLP's. The task is to train on the 194 I/O pairs until the learning system can produce the correct output for all of the inputs and then calculate the response for different inputs. Result of tests performed on 10200 equally spaced points in [-1,+1]×[-1,+1] is illustrated in Figure 3. Clearly the proposed architecture outperform the other network structure as regard regularization capability.

NET	N_l	*-20dB*			*-22dB*			*-24dB*			*1000 cycles*	
		<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	338	-10.90	1.58×10^{-3}	532	-11.51	1.39×10^{-3}	-	-	-	-11.64	1.26×10^{-3}
N_10	41	217	-10.60	2.98×10^{-3}	281	-10.76	4.32×10^{-3}	515	-10.52	5.31×10^{-3}	-12.32	9.22×10^{-3}
N_15	61	111	-10.88	2.31×10^{-3}	175	-11.82	1.31×10^{-3}	381	-12.26	9.98×10^{-4}	-13.21	4.76×10^{-3}
S1_2	17	188	-15.12	2.65×10^{-7}	-	-	-	-	-	-	-15.68	3.46×10^{-4}
S1_4	33	66	-14.79	9.07×10^{-5}	221	-15.98	1.79×10^{-5}	-	-	-	-16.22	1.17×10^{-5}
S2_1	24	47	-16.82	1.87×10^{-5}	176	-17.22	1.66×10^{-4}	294	-18.24	1.28×10^{-4}	-20.32	1.88×10^{-4}
S2_2	60	26	-15.32	2.42×10^{-4}	79	-17.12	3.78×10^{-4}	123	-16.35	1.06×10^{-3}	-19.89	1.24×10^{-3}

Table 1. Simulation results for fitting function (27).

NET	N_l	*-20dB*			*-22dB*			*-24dB*			*1000 cycles*	
		<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_11	45	63	-14.02	2.60×10^{-4}	599	-17.34	5.45×10^{-5}	-	-	-	-17.80	6.09×10^{-5}
N_15	61	48	-13.31	4.15×10^{-4}	414	-17.53	4.60×10^{-5}	-	-	-	-16.90	3.47×10^{-4}
S1_4	33	66	-17.85	7.94×10^{-6}	270	-19.68	2.09×10^{-5}	-	-	-	-13.94	1.26×10^{-4}
S2_1	24	47	-16.85	1.23×10^{-5}	264	-20.11	1.89×10^{-5}	478	-17.23	1.56×10^{-3}	-16.23	2.75×10^{-3}
S2_2	60	63	-18.34	3.45×10^{-5}	115	-21.36	7.18×10^{-5}	310	-20.19	2.98×10^{-4}	-17.84	3.45×10^{-4}

Table 2. Simulation results for fitting function (28).

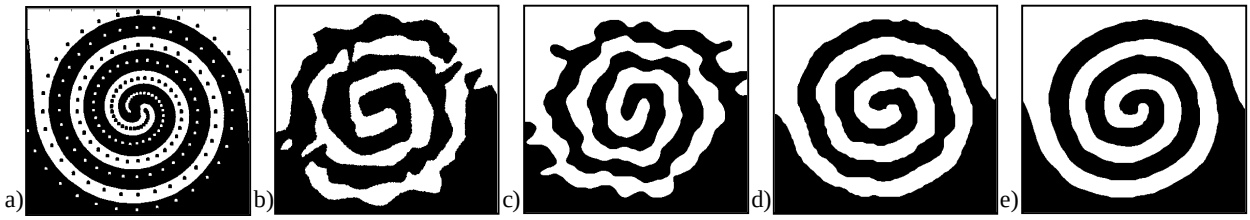

Figure 3. Two-spiral experiment results. a) Training data set (194 samples). b)-e) Resultant mapping for 10200 samples equally spaced in [-1,+1]× [-1,+1]: b) *N_5-5-5* (3 hidden layers), c) *S1_8*, d) *S2_2* and e) *S2_4*.

VII. REFERENCES

[1] Chen C.T., Chang W.D., "A Feedforward Neural Network with Function Shape Autotuning", *Neural Networks*, Vol.9, No 4, pp. 627-641, June 1996.

[2] Piazza F., Uncini A. and Zenobi M., "Neural Networks with Adaptive Polynomial Activation Function" *Proc. of the IJCNN Int. Joint Conf. on Neural Networks*, Beijing, China, vol.2 pp.343-349, Nov. 1992.

[3] Guarnieri S., Piazza F. and Uncini A., "Multilayer Neural Networks with Adaptive Spline-based Activation Functions", *Proc. of Word Congress on Neural Networks WCNN'95*, Washington D.C., USA, July 17-21, 1995.

[4] Vecci L., Campolucci P., Piazza F. and Uncini A., "Approximation Capabilities of Adaptive Spline Activation Function", *Proc. of International Conference on Neural Networks ICNN'97*, pp. 260-265, Houston TX, USA, Jun. 1997.

[5] Vecci L., Piazza F. and Uncini A., "Learning and Approximation Capabilities of Adaptive Spline Activation Function Neural Networks", *Neural Networks*, vol.11, No. 2, pp.259-270, Mar. 1998.

[6] Vitushkin A.G. and Henkin G.M., "Linear Superposition of Functions", *Russian Math. Surveys*, vol. 22 pp. 77-125, 1967.

[7] Tikhonov A.N., "Solution of Incorrectly Formulated Problems and the Regularization Method", *Soviet Math. Dokl.*, vol.4, pp. 1035-1038, 1963.

[8] Tikhonov A.N. and Arsenin V.Y., "Solution of Ill-Posed Problems", *W.H. Winston*, Washington, DC, 1977.

[9] Girosi F., Jones M. and Poggio T., "Regularization Theory and Neural Networks Architectures", *Neural Computations*, vol.7, pp. 219-269, 1995.

[10] Foley J.A. and Van Dam A., "*Fundamentals of Interactive Computer Graphics*", Addison-Wesley, ISBN 0-201-14468-9, 1982.

[11] Lang K. and Witbrock, "Learning to tell two spiral apart", *in Proc. 1988 Connectionist Models Summer School*. San Mateo, CA: Morgan Kaufmann, June 17-26, 1988, pp. 52-59.