Multilayer Neural Networks with Adaptive Spline-based Activation Functions

S. Guarnieri, F. Piazza, A. Uncini

Dip. di Elettronica e Automatica, Università di Ancona
via Brecce Bianche - 60131 Ancona, Italy.
Phone: +39 71 220 4453 - FAX: +39 71 220 4464
e-mail: upfm@ealab.unian.it

ABSTRACT

The importance of the activation functions on the behaviour of a MLP network is undeniable. Although such functions are usually sigmoidal, other functions, also depending on some free parameters, have been studied and applied. Most approaches tend to use relatively simple functions (as adaptive sigmoid), primarily due to computational complexity and difficulties in hardware implementation. In this paper a new adaptive activation function, built as a piecewise approximation with suitable cubic splines, is proposed. Such functions can have arbitrary shape, are efficiently implementable with digital hardware and allow to reduce the overall size of the neural networks, trading connection complexity with activation function complexity.

INTRODUCTION

It is well known that the classical neuron computes the weighted sum of the inputs and applies to this sum a non-linear function called activation function. The behaviour of a neural network built with such neurons, as the Multi Layer Perceptron (MLP) [1], thus depends on the chosen activation functions. Usually such functions are sigmoidal [1]; however, other functions, also depending on some free parameters, have been studied and applied [4,5,6]. The behaviour of different classes of non-linear activation functions with respect to the computational capabilities of the MLP has also been widely studied (see for example [2]).

Recently the use of polynomial functions has been proposed [4,5], in particular it has been shown that networks composed by polynomial neurons are isomorphic to conventional polynomial discriminant classifiers or Volterra filters, thus keeping their approximation capabilities. Moreover, such activation functions allow to reduce the size of the network, trading connection complexity with activation function complexity. When digitally implemented, the overall complexity of the resulting MLP can be reduced since the computation of the activation function does not depend heavily on the shape of the function, being usually performed through a Look-Up Table (LUT).

Networks with polynomial activation functions, however, cannot be universal approximators, as shown in [3]; moreover, such networks can have problems with spurious minima and maxima when their coefficients are adapted during the learning phase [4]. In [6] the direct adaptation of the LUT coefficients is proposed; while the reported experiments show a large reduction of the network size, nevertheless the adaptation process can be difficult mainly due to the large number of adaptive parameters and the loss of smoothness of the resulting functions.

The main idea of this work is to obtain an adaptive activation function for each neuron, which is squashing and smooth, by a piecewise approximation with suitable cubic splines. The use of the Catmull-Rom interpolation scheme [7] allows to efficiently implement both the forward and learning phase of the MLP. Several experimental results shows the computational capabilities of the proposed approach and the attainable network size reductions.

THE SG ACTIVATION FUNCTION

It has been demonstrated in [3] that a network with polynomial spline activation functions is an universal approximator. Such splines are, generally, smooth parametric curves, divided in multiple spans; they have also the property of preserving the continuity of the derivatives at the joining points. In the planar case the graph \( F_x(u) \) of the \( i \)-th curve span is represented by

\[
F_i(u) = [F_x(u) \ F_y(u)]^T
\]

(1)

where \( u \) is the parameter (usually varying between 0 and 1), \( T \) is the transpose operator and the two polynomial functions \( F_x(u), F_y(u) \) describe the curve span behaviour in the two coordinates \( x \) and \( y \).

As the designer has little or no control on the shape of a polynomial curve, it is preferable to switch to a representation
that makes use of the so-called spline basis functions: in this way the $i$-th curve span can be written in the form

$$F_i(u) = \sum_{k=0}^{d} Q_{i+k} b_{k,i}(u) = \begin{bmatrix} \sum_{k=0}^{d} q_{k,i+k} b_{k,i}(u) \\ \sum_{k=0}^{d} q_{j,i+k} b_{j,i}(u) \end{bmatrix}$$

(2)

where $b_{k,i}(u)$ is the $i$-th element of the spline basis (a polynomial of degree $d$ in the variable $u$) with $u \in [0,1]$, and $Q_{i+k} = [q_{i,j+k} \; q_{j,i+k}]^T$ are the $(d+1)$ control points of the $i$-th curve span: moving such points on the real plane will affect the shape of the curve.

Such spline schemes are called local schemes, as the shape of the curve in the $i$-th span is affected only by its $(d+1)$ control points, so that the curve can be modified locally without influencing distant spans.

The actual shape of the curve depends on the formulation of the basis functions: usually the base is chosen to let the curve lie "near" its control points. Moreover, such curve enjoys the variation diminishing property, that ensures the absence of unwanted oscillations between two consecutive control points, and the exact representation of linear segments.

Between the many spline bases presented in literature, we choose the Catmull-Rom cubic spline base [7], mainly for the following reasons:

- the control points lie on the curve itself, so that the obtained spline curve is an interpolator (see Fig. 1);
- the polynomial degree is $d = 3$, which means a good trade-off between curve complexity and calculations requirements;
- the curve’s first derivative is continuous at the control points.

Using the Catmull-Rom spline base, the $i$-th curve span in (2) can be written as

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

(3)

with $u \in [0,1]$. From (3) it is clear that every span is controlled by four control points: the second and the third will be interpolated, while the geometric tangent in these two points will be parallel to the segment passing through their adjacent control points as in Fig. 1

![Fig. 1](image)

A Catmull-Rom-based spline curve has a continuous first derivative (but not second); it is then possible to develop a backpropagation-style learning algorithm for neural networks that use such an activation function.

After having chosen the spline curve, we want to use it as an activation function in a neural network; this leads to the following two problems: a) how to choose the control points, and b) we need a function, not a plane curve.

The point b) is crucial: first we need to ensure that the curve is actually a function; then, given an abscissa value $x_0$, from (1) we calculate $u = F_x^{-1}(x_0)$, which is substituted in $F_{a}(\cdot)$ to deliver the corresponding value $y_0$ on the curve.

As $F_{a}(u)$ is a cubic polynomial, we need to solve a third degree equation (or use a numerical algorithm) to calculate $u$. The solution of this problem lies in the choice of the control points: if they are uniformly spaced along the x-axis (by a fixed step $\Delta x$), the variation diminishing property tells us that $F_{a}(u)$ will not be a cubic polynomial anymore: it will be a first degree polynomial.
Precisely, it is

$$F_a(u) = u \Delta x + q_{x,i+1}$$  \hspace{1cm} (4)$$

where $q_{x,i+1}$ is the abscissa value of the second control point of the $i$-th span ($Q_{i+1}$). For this reason we opted for the following choice of the spline control points: we take the sigmoidal function

$$F(x) = \frac{1}{1 + e^{-x}}$$  \hspace{1cm} (5)$$

and, fixed a step $\Delta x$ and an integer number $n$, we sample the function in $(n+1)$ uniformly spaced points along the $x$-axis ($Q_0, \ldots, Q_n$), in an interval centred on the axis origin (i.e. the sampling interval middle point lies on $(0,0)$). Outside the sampling interval the spline curve will have a constant value: $q_{x,l}$ for the negative $x$ coordinate, and $q_{x,r}$ for the positive $x$; also the first derivative will be constant outside the sampling interval: $(q_{x,l} - q_{x,0})/\Delta x$ and $(q_{x,r} - q_{x,0})/\Delta x$ respectively for the negative and positive $x$ coordinate.

This four points will never get modified, in order to maintain sigmoidal properties: we call this function SG function (acronym for the Italian "Sigmoida Generalizzata"). This choice ensures also that a SG function will always be a sufficiently kinky spline (in the sense explained in [31]): such a neural network is then an universal approximator.

As an implementation note, it is clear that the control points' abscissas are completely defined by the above procedure: this means that it is only necessary to store the $(Q_{0}, \ldots, Q_{n})$ values on a LUT to represent the SG function.

**THE SG NEURON**

The choice of a parametric curve as an activation function implies that, given a certain output value from the weighted sum, we need first to find its correspondence in the spline curve (i.e. which span it belongs and its parameter value), and then map the results to give the actual neuron output.

A neuron with a SG activation function (SG neuron) is shown in Fig. 2. Following the formalism of [1], it describes the $k$-th neuron in the $l$-th layer, where now the two SG blocks are needed in order to cope with the parametric functions (2). The block SG1 performs the inversion of the $x$-axis parametric function; due to the uniform sampling along this axis, the inversion problem can be carried out without solving a third order polynomial, but simply through the following steps:

$$z_{i}^{(0)} = \frac{x_{i}^{(0)} - n}{\Delta x} + 2 \quad a_{i}^{(0)} = \lfloor z_{i}^{(0)} \rfloor \quad u_{i}^{(0)} = z_{i}^{(0)} - a_{i}^{(0)}$$  \hspace{1cm} (6)$$

where $\lfloor \rfloor$ is the floor operator, while $a_{i}^{(0)}$ and $u_{i}^{(0)}$ are respectively the index of the involved patch and the value of the parameter $u$. The $z_{i}^{(0)}$ variable is an internal quantity.

![Fig. 2](image)

The block SG2 implements the formula (3) using the output of the first block, so that

$$x_{i}^{(0)} = F_{x,i}^{(0)}(u_{i}^{(0)})$$  \hspace{1cm} (7)$$

The function $F_{x,i}^{(0)}$ is now the $a_{i}^{(0)}$-th patch of the activation spline curve of the $k$-th neuron of the $l$-th layer. To the same neuron the sampling points $Q_{i}^{(0)}$ also belong. Note that the operations involved in both (6) and (7) are simple and suitable for a digital hardware implementation.

**THE LEARNING ALGORITHM**

The proposed learning algorithm is based on the classical BP, where the learning has been extended to the activation
function sampling points \( Q_{ij}^{(l)} \). For the \( i \)-th learning pattern, we will have

\[
\begin{align*}
\left[ w_{ij}^{(l)}(t+1) & = w_{ij}^{(l)}(t) + \Delta w_{ij}^{(l)}(t) \\
q_{k,(a,p\pm m)}^{(l)}(t+1) & = q_{k,(a,p\pm m)}^{(l)}(t) + \Delta q_{k,(a,p\pm m)}^{(l)}(t)
\end{align*}
\]  

(8)

with \( m = 0,\ldots,3 \) (from now on the time \( t \) will be omitted for the sake of readability), and \( \Delta \) represents the approximation of the gradient. The gradient-based approach requires the calculation of the first derivative of the activation function: the proposed spline curve ensures a continuous first derivative of SG1 and SG2 blocks.

The formula (3) states that the \( i \)-th span of the spline curve depends only on the \( u \) value: this is true unless the control points get modified, in which case such span would be a function of them also. Since the weight update must be performed with the activation function not yet adapted, we will consider the function univariate as in (3), so that only the derivative of the block SG2 with respect to the parameter \( u \) will be calculated. The derivative of the block SG1 is trivial to calculate since it is constant, as we expected from the variation diminishing property, and its value is \( \Delta x^1 \). This leads to the learning algorithm (always in the formalism of (1)):

\[
\begin{align*}
\varepsilon_k^{(l)} & = \begin{cases} 
\frac{ (d_k - x_k^{(l)}) }{ \sum_{p=1}^{N_{p}} \delta_k^{(l+1)} w_{k,p}^{(l+1)} } & l = M \\
\frac{ dF^{(l)}_{k,(a,p)}(u) }{ du } \bigg|_{u=a_k^{(l)}} \frac{1}{ \Delta x } & l = M - 1, \ldots, 1
\end{cases} \\
\delta_k^{(l)} & = \frac{ dF^{(l)}_{k,(a,p)}(u) }{ du } \bigg|_{u=a_k^{(l)}} \\
\Delta w_{ij}^{(l)} & = \mu \delta_k^{(l)} x_j^{(l-1)}
\end{align*}
\]  

(9)

with \( 0 \leq k \leq N_i \), \( 0 \leq j \leq N_{i-1} \) and where \( d_k \) is the desired \( k \)-th output value, and the derivative is a 2nd order polynomial.

For what concerns the control point adaptation, we need to recall the Catmull-Rom spline properties, particularly the locality of the interpolation approach: the parameter \( m \) ranging from 0 to 3 reflects this property, restricting the update to the \( a_k^{(l)} \) span, which involves 4 points. In this step we consider the parameter value fixed (i.e. \( u = a_k^{(l)} \)), so the SG2 block will be represented by a function of four variables (the four control points), resulting in the expression

\[
\Delta q_{k,(a,p\pm m)}^{(l)} = \mu_q \varepsilon_k^{(l)} \left( \frac{ \partial C^{(l)}_{k,(a,p\pm m)} }{ \partial a_k^{(l)} } \right) = \mu_q \varepsilon_k^{(l)} c_{k,m}^{(l)}(u_k^{(l)})
\]  

(10)

where \( m = 0,\ldots,3 \) and \( \mu_q \) is the learning rate for the activation function. The term \( c_{k,m}^{(l)}(\cdot) \) is the \( m \)-th element of the Catmull-Rom spline base (the \( m \)-th column of the matrix in (3)) calculated for \( u = a_k^{(l)} \). Formula (10) shows that only four control points at a time will be adapted, regardless of how many points are needed to describe the activation function: this means that the learning algorithm has a constant computational effort for each step.

**EXPERIMENTAL RESULTS**

We tested the proposed network on several problems in order to show the size reduction and its generalization properties, as well as the efficiency of the learning phase. For the sake of brevity, only two classical cases will follow. In every test the initial weight values are random, while each SG activation function has 28 control points, initially sampled from a sigmoid (with \( \Delta x = 0.6 \)).

**Case a:** 8 bit parity

Our approach is to use a network composed by a single SG neuron, and adapting the parameters with the learning rates \( \mu = 0.01 \) and \( \mu_q = 0.05 \). Fig. 3 shows the Mean-Square-Error (MSE) during the first 200 epochs. The shape of the SG activation function obtained at the end of this learning is reported in Fig. 4. Such a shape is reached approximately during the first 10-20 epochs, and then it does not change appreciably.
Case b: A-B region

This problem has been solved with a SG network 2-2-1 (2 inputs, first layer with 2 neurons and single neuron output layer); we make a comparison with a sigmoidal MLP network of dimensions 2-8-1. Using $\mu=0.01, \mu_0=0.05 (\mu=0.01$ for the sigmoidal network) and a learning set of 200 random points equally subdivided between the two regions, the SG network has been trained for 200 epochs (400 epochs for the sigmoidal network). The resulting MSEs, averaged on 20 different networks, are reported in Fig. 5 (continuous line for SG network, dashed line for classical MLP), while the generalization performance of the SG network (computed using a uniform grid of 40,000 points and an output threshold level of 0.9) is shown in Fig. 6.

References