Strict Interpolation of a Smooth Function and Its First Derivative Using a Linearly-Trained Radial Basis Function Neural Network

J.S.C. Prentice

Department of Applied Mathematics, University of Johannesburg

P.O. Box 524, Auckland Park, 2006, South Africa

Tel: 27-115-593-145 E-mail: jprentice@uj.ac.za

Abstract

We present a neural network, based on Gaussian functions, for interpolating a univariate function and its first derivative. The network is linearly trained, and constitutes a continuous piecewise approximation. It is based on the superposition of three standard Gaussian-based radial basis function networks. Analysis indicates that this network is a better approximation than the standard network.

Keywords: Radial basis function, Neural network, Approximation, Strict interpolation, Gaussian, First derivative

1. Introduction

Neural network approximation of continuous functions typically involves training algorithms that use discrete values of the function only, rather than its first derivative, in addition. In this paper, we suggest a linear algorithm for training a network to approximate a smooth function, using values of the function and its first derivative at discrete points. In a sense, we are trying to reproduce the idea of Hermite polynomial interpolation in the context of neural network approximation.

In the next section, we briefly describe standard radial basis function neural network interpolation. Thereafter, we describe our algorithm, discuss its stability present an error analysis, consider the possibility of optimizing the network, and present some numerical examples.

2. Standard Radial Basis Function Neural Network interpolation [Bishop, 2000; Haykin, 1999]

Let $f:\mathbb{R}\to\mathbb{R}$ be a continuous function. Assume that $(x_i, f(x_i))\,, i = 1, ..., N$ are available, where the nodes x_i are unique. The data $(x_i, f(x_i))$ are used to construct an approximation of the form

$$R(x) = \sum_{i=1}^{N} w_i \phi_i(x) \tag{1}$$

where the w_i are referred to as *weights* and the ϕ_i are N linearly independent radially symmetric basis functions (also called *neurons*). The linear combination above is called a radial basis function neural network (RBFNN). The most commonly used basis functions are *Gaussians*

$$\phi_i(x) = \exp\left(-a_i^2 (x - c_i)^2\right) \tag{2}$$

where $a_i > 0$ is the *width parameter* and c_i is the *center*. Note that

$$a_i^2 = \frac{1}{2\sigma_i^2} \tag{3}$$

where σ_i is the *standard deviation* of the Gaussian. Often, σ_i is chosen as the minimum of or some average of

$$\{(x_i - x_{i-1}), (x_{i+1} - x_i)\}$$

so that, if the nodes are equispaced, a_i is the same for each basis function. In this work we will consider the width parameter to be an adjustable parameter whose value will be determined subject to an optimizing condition.

Determination of the weights is referred to as *training* the network. In the case of *strict interpolation* (sometimes known as *exact* interpolation), we require that $R(x_i) = f(x_i)$ at each of the available nodes x_i. So, if there are N distinct nodes we have

$$\underbrace{\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_N(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & & \phi_N(x_2) \\ \vdots & & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_N(x_N) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}}_{W} = \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}}_{F} \tag{4}$$

in which the interpolation matrix Φ, the weight vector W and the target vector F have been defined. Solvability of this system depends on the determinant of the interpolation matrix; it has been shown that Φ is invertible for several types of basis function, including Gaussians [Michelli, 1986], provided that the nodes are distinct (for non-strict interpolation, when the number of nodes is different to the number of basis functions, the matrix Φ is not square and the linear system is solved by taking the pseudoinverse of Φ).

The strict interpolation problem described above is thus a straightforward one - invertibility is ensured for Gaussian basis functions, and solving linear systems computationally is easily accomplished using modern numerical software.

Our objective is to design a strict interpolation RBFNN that is trained linearly and that also interpolates the first derivative of $f(x)$, in the spirit of Hermite polynomial interpolation [Mhaskar & Pai, 2000]. Of course, one expects, as in the case of Hermite interpolation, that interpolation of the first derivative, in addition to the function itself, will result in a more accurate approximation. To the best of our knowledge, this has not been attempted before, and so constitutes a gap in the body of research in this field. It is our intention to attempt to fill this gap. Our algorithm, designated the *superposition network* (SPN), is described in the next section.

3. Description of the SPN algorithm

Assume that $\left\{x_i, f_i, f_i'\right\}, i = 1, ..., N$ are available, where $f_i \equiv f(x_i)$ and $f_i' \equiv f'(x_i)$. An algorithm for constructing an RBFNN approximation to $f(x)$ on a subinterval $[x_i, x_{i+1}]$, which interpolates $(x_i, f_i), \left(x_i, f_i'\right), (x_{i+1}, f_{i+1})$ and $\left(x_{i+1}, f_{i+1}'\right)$ proceeds as follows:

1. Choose a width parameter a_i by simply taking σ_i to be the average spacing h_{ave} between the nodes on $[x_{i-1}, x_{i+2}]$. We then have $a_i^2 = \frac{1}{2h_{ave}^2}$.

2. Construct three standard RBFNNs, using width parameters $\frac{a_i}{\sqrt{2}}$, a_i and $\sqrt{\frac{3}{2}}a_i$. These networks each interpolate (x_i, f_i) and (x_{i+1}, f_{i+1}). The number of neurons in each network should be at least two, although we will use four in this work, centered at the nodes $\{x_{i-1}, x_i, x_{i+1}, x_{i+2}\}$. We will refer to these networks as *component networks*, and we denote them by R_1, R_2 and R_3. These component networks are constructed on the subinterval $[x_{i-1}, x_{i+2}]$ and each of them interpolates $\{f_{i-1}, f_i, f_{i+1}, f_{i+2}\}$, hence the use of $[x_{i-1}, x_{i+2}]$ to determine the width parameter a_i. It is important to note that the guaranteed inversion of Φ ensures that these three component networks do exist.

3. We now determine
$$\Delta_p^v \equiv f'(x_v) - R_p'(x_v) \tag{5}$$
for $v \in \{i, i+1\}$ and $p = 1, 2, 3$. These Δs are the differences between the derivatives of the component networks and the objective function $f(x)$ at the nodes x_i and x_{i+1}, and we will refer to them as *derivative residuals*.

4. We now seek α, β and γ such that
$$\begin{aligned} \alpha + \beta + \gamma &= 1 \\ f'(x_i) - \alpha R_1'(x_i; a_i) - \beta R_2'(x_i; a_i) - \gamma R_3'(x_i; a_i) &= 0 \\ f'(x_{i+1}) - \alpha R_1'(x_{i+1}; a_i) - \beta R_2'(x_{i+1}; a_i) - \gamma R_3'(x_{i+1}; a_i) &= 0 \end{aligned} \tag{6}$$
which gives
$$\begin{aligned} &(\alpha + \beta + (1 - \alpha - \beta)) f'(x_i) \\ &-\alpha R_1'(x_i; a_i) - \beta R_2'(x_i; a_i) \\ &-(1 - \alpha - \beta) R_3'(x_i; a_i) = 0 \end{aligned} \tag{7}$$
$$\begin{aligned} &(\alpha + \beta + (1 - \alpha - \beta)) f'(x_{i+1}) \\ &-\alpha R_1'(x_{i+1}; a_i) - \beta R_2'(x_{i+1}; a_i) \\ &-(1 - \alpha - \beta) R_3'(x_{i+1}; a_i) = 0. \end{aligned} \tag{8}$$
Hence,
$$\alpha\Delta_1^i + \beta\Delta_2^i + (1 - \alpha - \beta)\Delta_3^i = 0 \tag{9}$$
$$\alpha\Delta_1^{i+1} + \beta\Delta_2^{i+1} + (1 - \alpha - \beta)\Delta_3^{i+1} = 0 \tag{10}$$
which may be written as
$$\underbrace{\begin{bmatrix} \Delta_1^i - \Delta_3^i & \Delta_2^i - \Delta_3^i \\ \Delta_1^{i+1} - \Delta_3^{i+1} & \Delta_2^{i+1} - \Delta_3^{i+1} \end{bmatrix}}_{T} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} -\Delta_3^i \\ -\Delta_3^{i+1} \end{bmatrix}. \tag{11}$$

5. If we now form the linear combination

$$R_{i,i+1}(x;a_i) \equiv \alpha R_1(x;a_i) + \beta R_2(x;a_i) + (1-\alpha-\beta)\, R_3(x;a_i) \tag{12}$$

then $R_{i,i+1}(x;a_i)$ is a superposition of standard RBFNNs, and $R_{i,i+1}(x)$ interpolates (x_i, f_i), $\left(x_i, f_i'\right)$, (x_{i+1}, f_{i+1}) and $\left(x_{i+1}, f_{i+1}'\right)$. We write $R_{i,i+1}(x;a_i)$ since $R_{i,i+1}(x;a_i)$ is valid only on $[x_i, x_{i+1}]$. Our notation also emphasizes the dependence of $R_{i,i+1}, R_1, R_2$ and R_3 on the parameter a_i.

6. For the subintervals at the extreme left and extreme right of the interval, $[x_1, x_2]$ and $[x_{N-1}, x_N]$, the same procedure applies except that we determine the derivative residuals at $\{x_1, x_2\}$ and $\{x_{N-1}, x_N\}$, respectively.

7. We apply the above procedure on each subinterval, so generating $N-1$ networks $R_{i,i+1}, i = 1, ..., N-1$. If we now define

$$\Theta_i(x) \equiv \begin{cases} R_{i,i+1}(x;a_i) & x \in [x_i, x_{i+1}] \\ 0 & x \notin [x_i, x_{i+1}] \end{cases},$$

then the approximation $R_{SPN}(x)$ on the entire interval $[x_1, x_N]$ is simply

$$R_{SPN}(x) \equiv \sum_{i=1}^{N-1} \Theta_i(x).$$

Note that, since $R_{i-1,i}$ and $R_{i,i+1}$ each interpolate (x_i, f_i) and $\left(x_i, f_i'\right)$, we have that $R_{SPN}(x)$ is differentiable.

8. Note that since R_{SPN} is a linear combination of standard RBFNNs, any convergence results that hold for a standard RBFNN [Williamson, 1995; Liu & Si, 1994; Park & Sandberg, 1991] also hold for R_{SPN}.

9. **Comments:** The choice of width parameters $\frac{a_i}{\sqrt{2}}, a_i$ and $\sqrt{\frac{3}{2}}a_i$ for the component networks is somewhat arbitrary. Whatever values are used, they must be such that the derivative residuals for the component networks are not identical - otherwise we run the risk of having a row or column of zeros in the coefficient matrix T in Step 4 (see next section). Also, the number of neurons in each component network could be as high as N. We do not need to base the approximation on subinterval approximation; it is easy to extend this idea to a superposition of $N+1$ networks, each of which interpolates $f(x)$ on the entire set (x_i, f_i), although the resulting coefficient matrix T is large ($N \times N$). The virtue of a 2×2 coefficient matrix T is that it is potentially easier to ensure invertibility (see next section). From a stability point of view, the inversion of T could present a problem if it is not well-conditioned. Our approach in this work is subinterval (piecewise) approximation.

4. Stability

4.1 Invertibility of the matrix Φ

In consideration of the stability of SPN, we note that as $a_i \to 0$, Φ tends to the unit matrix, which is singular. However, as mentioned earlier, for nonzero width parameters Φ is invertible. Nevertheless, we should be careful not to choose width parameters too small, lest Φ becomes ill-conditioned.

In a 'direct' interpolation algorithm, the interpolation matrix would contain the first derivative of the Gaussians, in addition to the Gaussians themselves. As far as we are aware, no guarantee can be given for the invertibility of such a matrix; this is one of our prime motivations for developing SPN, in which we can rely on the guaranteed invertibility of Φ.

4.2 Invertibility of the matrix T

We have

$$\begin{aligned} T &= \begin{bmatrix} \Delta_1^i - \Delta_3^i & \Delta_2^i - \Delta_3^i \\ \Delta_1^{i+1} - \Delta_3^{i+1} & \Delta_2^{i+1} - \Delta_3^{i+1} \end{bmatrix} \\ &= \begin{bmatrix} R_3'(x_i) - R_1'(x_i) & R_3'(x_i) - R_2'(x_i) \\ R_3'(x_{i+1}) - R_3'(x_{i+1}) & R_3'(x_{i+1}) - R_2'(x_{i+1}) \end{bmatrix}. \end{aligned} \tag{13}$$

As $a_i \to \infty$, $T \to \mathbf{0}$, since the derivative of a Gaussian at its center is zero, and the derivative of a Gaussian at any other point tends to zero as $a_i \to \infty$. This means that the entries of the superposition matrix T tend to zero and so T becomes singular. Also, as $a_i \to 0$, the Gaussians become horizontal on $[x_i, x_{i+1}]$, in which case all derivatives in T are zero.

There is a special case that must be considered. If the objective function is symmetric or antisymmetric on the interval $[x_{i-1}, x_{i+2}]$, and the nodes $\{x_{i-1}, x_i, x_{i+1}, x_{i+2}\}$ are uniformly distributed, then the component networks R_1, R_2 and R_3 will also be symmetric or antisymmetric (because the Gaussian basis functions are symmetric). Examples of such functions are $\sin(x)$ on $[0, \pi]$ (symmetric) and $\sin(x)$ on $[0, 2\pi]$ (antisymmetric). For the antisymmetric case we would have $f_i' = f_{i+1}'$ and $R_p'(x_i) = R_p'(x_{i+1})$ so that

$$T = \begin{bmatrix} \Delta_1^i - \Delta_3^i & \Delta_2^i - \Delta_3^i \\ \Delta_1^i - \Delta_3^i & \Delta_2^i - \Delta_3^i \end{bmatrix} \tag{14}$$

which is singular. For the symmetric case we would have $f_i' = -f_{i+1}'$ and $R_p'(x_i) = -R_p'(x_{i+1})$ so that

$$T = \begin{bmatrix} \Delta_1^i - \Delta_3^i & \Delta_2^i - \Delta_3^i \\ -\left(\Delta_1^i - \Delta_3^i\right) & -\left(\Delta_2^i - \Delta_3^i\right) \end{bmatrix} \tag{15}$$

which is also singular. Naturally, the symmetry can be broken simply by choosing nodes that are not symmetrical about the centre of the interval.

However, if it is not possible to redefine the nodes, we can still offer a solution in the form of the linear system

$$\begin{bmatrix} 1 & 1 \\ \Delta_1^i - \Delta_3^i & \Delta_2^i - \Delta_3^i \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ -\Delta_3^i \end{bmatrix} \tag{16}$$

where the first equation comes from the condition

$$1 - \alpha - \beta = \frac{\alpha + \beta}{2}. \tag{17}$$

This condition is motivated by requiring that the coefficient of the network R_3 is the average of the coefficients of R_1 and R_2, although any other similar condition could be imposed. The coefficient matrix in this system has determinant

$$\det\left(\begin{bmatrix} 1 & 1 \\ \Delta_1^i - \Delta_3^i & \Delta_2^i - \Delta_3^i \end{bmatrix}\right) = \Delta_2^i - \Delta_1^i = R_1'(x_i) - R_2'(x_i). \tag{18}$$

We must show that this determinant is nonzero: assume

$$\begin{aligned} R_1(x) &= \sum_{j=1}^{4} w_j \exp\left(-a_1^2\left(x - c_j\right)^2\right) \\ \Rightarrow R_1'(x) &= \sum_{j=1}^{4} -a_1^2\left(x - c_j\right) w_j \exp\left(-a_1^2\left(x - c_j\right)^2\right) \qquad (19) \\ R_2(x) &= \sum_{j=1}^{4} z_j \exp\left(-a_2^2\left(x - c_j\right)^2\right) \\ \Rightarrow R_2'(x) &= \sum_{j=1}^{4} -a_2^2\left(x - c_j\right) z_j \exp\left(-a_2^2\left(x - c_j\right)^2\right). \qquad (20) \end{aligned}$$

Now,

$$R_1'(x_i) = \sum_{j=1}^{4} -a_1^2\left(x_i - c_j\right) w_j \exp\left(-a_1^2\left(x_i - c_j\right)^2\right) \tag{21}$$

$$R_2'(x_i) = \sum_{j=1}^{4} -a_2^2\left(x_i - c_j\right) z_j \exp\left(-a_2^2\left(x_i - c_j\right)^2\right) \tag{22}$$

and, since the Gaussian functions are linearly independent,

$$R_1'(x_i) - R_2'(x_i) = 0 \Rightarrow a_1^2\left(x_i - c_j\right) w_j = 0 \text{ and } a_2^2\left(x_i - c_j\right) z_j = 0 \tag{23}$$

for each j. Now, since w_j and z_j cannot be expected to be identically zero in all cases, we must have

$$a_1 = 0 \text{ and } a_2 = 0 \tag{24}$$

in order for (23) to be generally true. But the widths of the component networks are intentionally chosen to be nonzero. Thus, we have a contradiction and we conclude

$$R_1'(x_i) - R_2'(x_i) \neq 0. \tag{25}$$

We would resort to the system (16) only if we detected the presence of the type of (anti)symmetry problem described here.

4.3 Stability test

For brevity we will write $R(x;a_i)$ instead of $R_{i,i+1}(x;a_i)$ from now on. If either of the matrices T and Φ is ill-conditioned, SPN will fail to properly interpolate (x_i, f_i), $\left(x_i, f_i'\right)$, (x_{i+1}, f_{i+1}) and $\left(x_{i+1}, f_{i+1}'\right)$. Measuring $|f_i - R(x_i;a_i)|$, $\left|f_i' - R'(x_i;a_i)\right|$, $|f_{i+1} - R(x_{i+1};a_i)|$ and $\left|f_{i+1}' - R'(x_{i+1};a_i)\right|$ gives an indication of the stability of SPN for any given width parameter a_i. A tolerance on these values could be specified so that if this tolerance is breached, the corresponding a_i-value is rejected.

Indeed, we define the *end-point-conditions* (EPCs) by

$$\begin{aligned}\Gamma(a_i) &\equiv \frac{|R(x_i;a_i) - f_i|}{1+|f_i|} + \frac{|R(x_{i+1};a_i) - f_{i+1}|}{1+|f_{i+1}|} && (26)\\ \Gamma'(a_i) &\equiv \frac{\left|R'(x_i;a_i) - f_i'\right|}{1+\left|f_i'\right|} + \frac{\left|R'(x_{i+1};a_i) - f_{i+1}'\right|}{1+\left|f_{i+1}'\right|}. && (27)\end{aligned}$$

The denominators in these expressions contain a 1 in case f_i or f_{i+1}' are close to zero. These EPCs measure the extent to which SPN properly interpolates f_i, f_{i+1}, f_i' and f_{i+1}', as is intended.

5. Error analysis

We have that $R(x;a_i)$ approximates $f(x)$, such that $f = R$ at $\{x_{i-1}, x_i, x_{i+1}, x_{i+2}\}$ and $f' = R'$ at $\{x_i, x_{i+1}\}$. Hence, the function $\Omega(x;a_i) \equiv f(x) - R(x;a_i)$ has two roots of multiplicity one at $\{x_{i-1}, x_{i+2}\}$ and two roots of multiplicity two at $\{x_i, x_{i+1}\}$. So, we may write

$$\Omega(x;a_i) = \Upsilon(x;a_i)(x-x_i)^2(x-x_{i+1})^2(x-x_{i-1})(x-x_{i+2}) \tag{28}$$

where $\Upsilon(x;a_i)$ is a function to be determined.

Now, define the function $F(z)$ by

$$F(z) \equiv f(z) - R(z;a_i) - \Upsilon(x;a_i)(x-x_i)^2(x-x_{i+1})^2(x-x_{i-1})(x-x_{i+2}). \tag{29}$$

So $F(z) = 0$ at $\{x_{i-1}, x_i, x_{i+1}, x_{i+2}\}$ and at some $x \notin \{x_{i-1}, x_i, x_{i+1}, x_{i+2}\}$, a total of five distinct points in $[x_{i-1}, x_{i+2}]$. These five points define four adjacent subintervals, and somewhere on each subinterval there is at least one point such that $F'(z) = 0$ (by Rolle's theorem). Thus, there are four points distinct from the nodes at which $F'(z) = 0$, and $F'(z) = 0$ also at x_i and x_{i+1}. Hence, $F'(z) = 0$ at six distinct points. This means that $F''(z) = 0$ at five distinct points and so on, until we have that $F^{(6)}(z) = 0$ at one point in $[x_{i-1}, x_{i+2}]$, say $z = \zeta(x;a_i)$. We indicate that ζ can generally be expected to be dependent on x and a_i.

Therefore, we have

$$F^{(6)}(\zeta(x;a_i)) = 0 = f^{(6)}(\zeta(x;a_i)) - R^{(6)}(\zeta(x;a_i)) - 720\Upsilon(x;a_i) \tag{30}$$

since the sixth derivative of $(x-x_i)^2(x-x_{i+1})^2(x-x_{i-1})(x-x_{i+2})$ equals 720. Hence

$$\Upsilon(x;a_i) = \frac{\Omega^{(6)}(\zeta(x;a_i))}{720} \tag{31}$$

and so

$$\Omega(x;a_i) = \frac{\Omega^{(6)}(\zeta(x;a_i))}{720}(x-x_i)^2(x-x_{i+1})^2(x-x_{i-1})(x-x_{i+2}). \tag{32}$$

This is true for all x in $[x_{i-1}, x_{i+2}]$.

It is easily shown that

$$|\Omega(x;a_i)| \leqslant \left(\frac{\max\limits_{[x_{i-1},x_{i+2}]}\left|\Omega^{(6)}(\zeta(x);a_i)\right|}{180}\right) h_{max}^6 \tag{33}$$

which is a sixth-order bound, wherein h_{max} denotes the maximum separation of the nodes x_{i-1}, x_i, x_{i+1} and x_{i+2}. Note that we have assumed that $f(x)$ is suitably differentiable in this derivation; certainly, $R(x;a_i)$ is infinitely differentiable w.r.t. x since it is a linear combination of Gaussians.

Other upper bounds may be derived in a similar manner by assuming any of the following:

$$\begin{aligned}\Omega(x;a_i) &= \Upsilon(x;a_i)(x-x_i)^2\\ \Omega(x;a_i) &= \Upsilon(x;a_i)(x-x_{i+1})^2\\ \Omega(x;a_i) &= \Upsilon(x;a_i)(x-x_i)^2(x-x_{i+1})^2.\end{aligned} \tag{34}$$

The first two of these lead to second-order error expressions, and the last one leads to a fourth-order expression and, consequently, second- and fourth-order bounds. Clearly, these bounds are less stringent than the sixth-order bound in (33). It must be noted here that the standard four-neuron network on $[x_{i-1}, x_{i+2}]$ cannot have error better than $O\left(h_{max}^4\right)$, so we see that R_{SPN} is expected to be a more accurate approximation than the standard network.

6. Optimal a_i

The traditional approximation problem is to replace a given $f(x)$ with a linear combination of preselected basis functions. In such case, $f(x)$ is known explicitly and it is easy to measure the quality of $R(x; a_i)$. We simply determine the *cost function*

$$C(a_i) \equiv \frac{1}{M}\sum_{m=1}^{M} |f(x_m) - R(x_m; a_i)| \tag{35}$$

on $[x_i, x_{i+1}]$ at M nodes x_m, where M is large. We then determine $C(a_i)$ for a number of a_i-values in some interval $[a_{i\min}, a_{i\max}]$ and settle on that a_i-value which gives a minimum $C(a_i)$. This is, admittedly, a brute force approach, but it is effective in a numerical context. However, it must be noted that this approach is easy and fast, due to the linear character of SPN.

If $f(x)$ is not known explicitly, but rather only values of $f(x)$ are available at discrete nodes x_i, it is more difficult to estimate $C(a_i)$ reliably. If the f_i are close together it might be a good idea to choose $[x_i, x_{i+1}]$ such that this subinterval actually contains Q values of $f(x)$ at nodes x_q between x_i and x_{i+1}. These data points are not used in the construction of $R(x; a_i)$, but rather are used to determine

$$C(a_i) = \frac{1}{Q}\sum_{q=1}^{Q} \left| f\left(x_q\right) - R\left(x_q; a_i\right)\right|. \tag{36}$$

This idea of omitting some data for the purpose of quality control - termed *generalization* - is often used in neural networking. Other cost functions that could be used are

$$C(a_i) = \max |f(x_m) - R(x_m; a_i)| \tag{37}$$

$$C(a_i) = \sqrt{\frac{1}{M}\sum_{m=1}^{M} [f(x_m) - R(x_m; a_i)]^2}. \tag{38}$$

We will use both (35) and (37) in our examples in the next section.

7. Numerical examples

We have approximated $f(x) = e^x$ on $\left[\frac{4}{3}, \frac{5}{3}\right]$, $f(x) = \sin(x)$ on $\left[\frac{\pi}{6}, \frac{\pi}{3}\right]$, and $f(x) = x$ on $[0, 1]$. For $f(x) = e^x$ SPN was about three orders of magnitude better than the standard network; for $f(x) = \sin(x)$ SPN was about two orders better; and for $f(x) = x$ SPN was about three orders better.

Figure 1 shows the cost $C(a)$ defined in (37) for the $f(x) = e^x$ example, for both the standard RBFNN with four neurons (denoted STN from now on), and SPN. Clearly, there is a range of a-values for which SPN is considerably more accurate than the standard algorithm. The jagged effect for small a-values is due to the ill-conditioning of the interpolation matrix Φ. The quantity a_h indicates the heuristic width parameter given by $\frac{1}{\sqrt{2}h_{ave}}$ as described previously. We see that the optimal width parameter is different to a_h. Figure 1 is typical of the results obtained for the other numerical examples.

Figure 2 shows the EPCs for the $f(x) = e^x$ example. In this figure 'Function EPC' is $\Gamma(a_i)$, and 'Function derivative EPC' is $\Gamma'(a_i)$. We see that the EPCs become large for both small and large values of a, and that there is an intermediate range of a-values for which the EPCs are of the order of machine precision ($\sim 10^{-16}$).

We have also approximated the function

$$H(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6 \tag{39}$$

which has both a strong and a weak maximum on the interval $[0, 1]$. Our approach is to choose 100 evenly spaced nodes on $[0, 1]$, so that $[0, 1]$ is divided into 99 subintervals. We apply SPN and STN to find approximations on each subinterval, each optimized with respect to the width parameter a_i. Figure 3 shows the cost function (35) on each subinterval for STN and SPN. Clearly, SPN is better than STN everywhere, and generally by several orders. Figure 4 shows the optimal width parameters per subinterval for this example, for both STN and SPN, demonstrating that they can differ from subinterval to subinterval.

Lastly, we approximate $\sin x$ on $[0, \pi]$ at the equispaced nodes $\left\{0, \frac{\pi}{3}, \frac{2\pi}{3}, \pi\right\}$, using the heuristic width parameter, so as to illustrate the use of (16). Because of the symmetric character of $\sin x$ on this interval, and the uniform spacing of the nodes, the matrix T is singular. However, we find that the coefficient matrix in (16) has determinant of $-0.1114...$ and condition number ~ 18, and so is reliably inverted. Hence, we find that STN has a maximum error of 0.05, and SPN has a maximum error of 8×10^{-4}, on $\left[\frac{\pi}{3}, \frac{2\pi}{3}\right]$.

8. Conclusion

The use of RBF networks to approximate nonlinear functions, using a discrete set of values of both the function and its first derivative has, to the best of our knowledge, not been attempted before. In this paper, we have addressed this issue by developing a strict interpolation approximation algorithm based on Gaussian basis functions, that not only interpolates the target function but also its first derivative. The approximation is piecewise, and is based on a superposition of three standard RBF networks. Coefficients in the superposition are determined linearly, using a discrete set of values of both the function and its first derivative as training data. By implementing the algorithm in such a manner, we are able to exploit the guaranteed invertibility of the coefficient matrices associated with standard RBF networks. In the special case of approximating a symmetrical/antisymmetrical function, a variation in the training algorithm has been given that avoids potential singularities in the relevant coefficient matrix. The possibility of using cost functions to determine an optimal width parameter has been suggested. Stability of the algorithm has been investigated, and a simple test for instability, w.r.t. the width parameter, has been proposed. A theoretical derivation of the approximation error indicates that the algorithm can be expected to be sixth-order in the node spacing h, while the standard network is only fourth-order. Numerical examples demonstrate the superiority of the algorithm over the standard approach, as anticipated.

Acknowledgment

Part of this work was incorporated into a PhD thesis, awarded by the Rand Afrikaans University (now the University of Johannesburg). The author gratefully acknowledges the role of the thesis supervisor, Prof. W.-H. Steeb.

References

Bishop, C.M. (2000). *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press, p164.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*, New Jersey: Prentice-Hall, p256.

Liu, B., and Si, J. (1994). The best approximation to C^2 functions and its error bounds using regular-center gaussian networks. *IEEE Trans. on Neural Networks*, 5, 5, 845-847.

Michelli, C.A. (1986). Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constr. Approx.*, 2, 11-22.

Mhaskar, H.N., and Pai, D.V. (2000). *Fundamentals of Approximation Theory*, Boca Raton: CRC Press, p185.

Park, J., and Sandberg, I.W. (1991). Universal approximation using radial-basis-function networks. *Neural Computation,* 3, 246-257.

Williamson, R.C. (1995). Existence and uniqueness results for neural network approximations. *IEEE Trans. on Neural Networks*, 6, 1, 2-13.

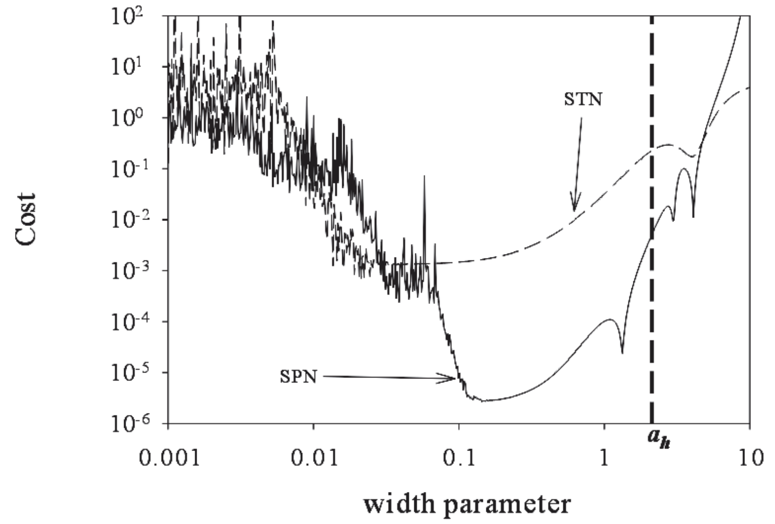

Figure 1. Cost function vs width parameter for STN and SPN when approximating $f(x) = e^x$.

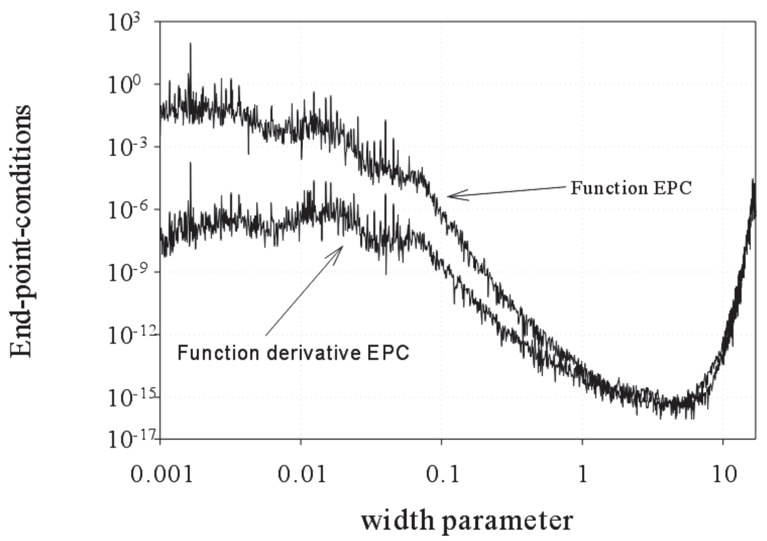

Figure 2. End-point-conditions (EPCs) vs width parameter for SPN when approximating $f(x) = e^x$

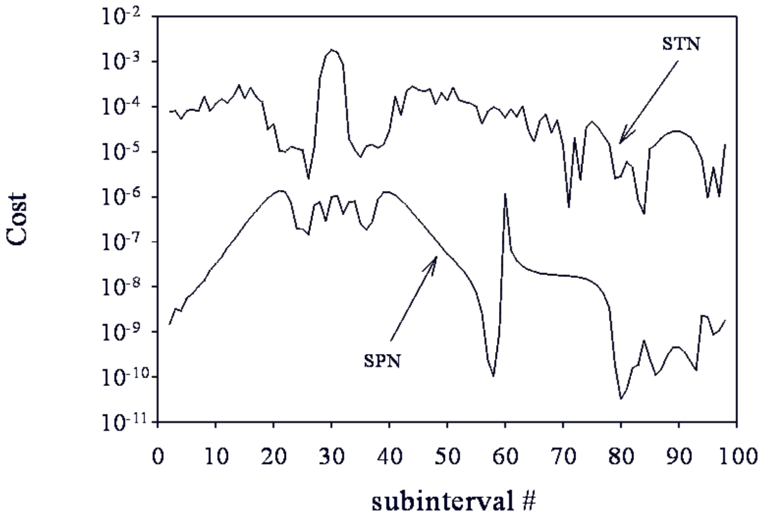

Figure 3. Cost function per subinterval for STN and SPN when approximating the function $H(x)$, as described in the text

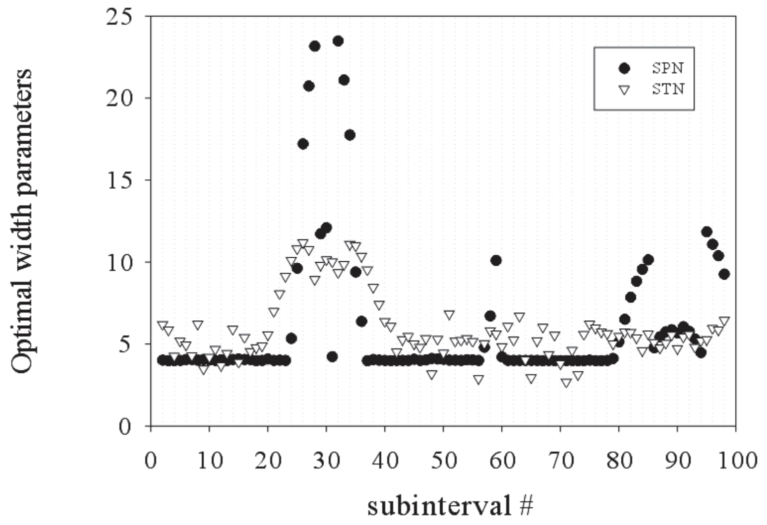

Figure 4. Optimal width parameters per subinterval for STN and SPN, when approximating $H(x)$