Neural Processing Letters (2019) 50:121–147
https://doi.org/10.1007/s11063-018-09974-4

Piecewise Polynomial Activation Functions for Feedforward Neural Networks

Ezequiel López-Rubio[1] · Francisco Ortega-Zamorano[1] · Enrique Domínguez[1] · José Muñoz-Pérez[1]

Published online: 10 January 2019

Abstract
Since the origins of artificial neural network research, many models of feedforward networks have been proposed. This paper presents an algorithm which adapts the shape of the activation function to the training data, so that it is learned along with the connection weights. The activation function is interpreted as a piecewise polynomial approximation to the distribution function of the argument of the activation function. An online learning procedure is given, and it is formally proved that it makes the training error decrease or stay the same except for extreme cases. Moreover, the model is computationally simpler than standard feedforward networks, so that it is suitable for implementation on FPGAs and microcontrollers. However, our present proposal is limited to two-layer, one-output-neuron architectures due to the lack of differentiability of the learned activation functions with respect to the node locations. Experimental results are provided, which show the performance of the proposal algorithm for classification and regression applications.

Keywords Activation functions · Feedforward neural networks · Supervised learning · Regression · Classification

1 Introduction

Nonlinear activation functions are required to introduce nonlinearity in neural networks. The reason is that any composition of linear functions is also linear. The nonlinearity of a

✉ Ezequiel López-Rubio
ezeqlr@lcc.uma.es

Francisco Ortega-Zamorano
fortega@lcc.uma.es

Enrique Domínguez
enriqued@lcc.uma.es
http://www.lcc.uma.es/∼enriqued

José Muñoz-Pérez
munozp@lcc.uma.es

1 Department of Computer Languages and Computer Science, University of Málaga, Bulevar Louis Pasteur, 35, 29071 Málaga, Spain

neural network, i.e., its ability to represent nonlinear functions, is what makes them powerful. The standard model of feedforward neural networks was established in [26]. These are the multilayer perceptrons with learning by error backpropagation. They require differentiable activation functions, such as the logistic function or the hyperbolic tangent. In most cases, neural network models employ parametric models for the activation function. This assumes that the shape of the function is perfectly known, except for some tunable parameter, as in the cases of the logistic and hyperbolic tangent functions.

Sometimes the activation function is chosen to be bounded in order to bound the range of the output of the processing unit (artificial neuron). For this reason, it is sometimes referred to as the squashing function [17]. Generally speaking, the output range is either (0, 1) or $(-1, 1)$. Monotone non-decreasing activation functions are selected for this purpose. The most popular choices are the logistic function [34], the hyperbolic tangent function and its generalization [8] or sigmoid functions [2,8–10]. All of them have little flexibility, with at most a shape parameter and a location parameter that can be adjusted during the training process [8]. Other proposed adaptive functions are based on additive spline models [32], Gaussian mixtures models [5,6] or linear regressions [11]. But these trainable activation functions proposed in the literature assume a parametric model, i.e., no matter how big the training set is, the number of trainable parameters of the activation function is fixed and small.

In this paper, a new feedforward artificial neural network model with one output neuron and one hidden layer is proposed. We call it the piecewise polynomial activation function neural network (PPAFNN). It features piecewise polynomial, monotone non-decreasing activation functions which approximate the distribution functions of the synaptic potentials. Their shapes are adapted to the training data in a nonparametric fashion, so that a different activation function is learned for each neuron. Therefore, the number of trainable parameters of each activation function can be enlarged for big training set sizes. Our proposal is based on the estimation of some quantiles of the synaptic potential of each neuron, which are associated with the nodes of the piecewise activation function. In this way, the activation function is adapted to approximate the distribution function of the synaptic potential. In doing so, each neuron captures the specificity of the distribution of the incoming synaptic potentials in a more detailed way than the previously employed parametric activation functions. For example, if most of the synaptic potentials fall into a flat region of a parametric activation function, then the neuron outputs roughly the same value for most inputs, i.e., the neuron does not carry out any useful computation. Our approach tries to avoid this situation by adapting the nodes of the activation function to the distribution of the synaptic potential, so that the neuron output values span the full range of the activation function. However, our approach is limited to small architectures with two layers and one output neuron. This is because the chain rule cannot be employed due to the lack of differentiability of the piecewise polynomial activation functions that we employ, with respect to the node locations, being this work a preliminary research of neural networks with this type of activation function. The main advantage of our approach, as compared to previous proposals of learnable activation functions, is that our approach does not require the calculation of any transcendental function. Therefore, it is particularly suitable for its implementation on microcontrollers and other low-cost devices which do not implement transcendental functions on hardware.

The structure of this paper is as follows: First, a review of related works is carried out in Sect. 2. After that, the proposed method is presented in Sect. 3. Experimental results are reported in Sect. 4. Finally, Sects. 5 and 6 are devoted to discussion and conclusions, respectively.

2 Previous Works

Next, we review the literature on activation functions for feedforward neural networks. One of the first proposals to adjust the activation function to the data is that of the generalized hyperbolic tangent function [8]:

$$g\left(u\right)=\frac{a\left(1-\exp\left(-bu\right)\right)}{1+\exp\left(-bu\right)} \tag{1}$$

where a and b represent the maximum value and the slope of g, respectively. This is a parametric model for the activation function, i.e., only a limited set of parameters can be adjusted (two in this case). Hence, the flexibility of the shape of g is small.

The amplitude of the activation function can be learned [30], so that the λ parameter is adjusted in:

$$g\left(u\right)=\lambda\tilde{g}\left(u\right) \tag{2}$$

where $\tilde{g}\left(u\right)$ does not depend on λ. Even if the nonlinearity $\tilde{g}\left(u\right)$ can be chosen at will, the number of trainable parameters is even smaller, since only one parameter is adjusted, which plays a role similar to that of a in (1).

The p-recursive piecewise polynomial sigmoid activation functions [29] aim to approximate the hyperbolic tangent with a lower computational effort. Hence, they stick to the shape of the hyperbolic tangent, with no adaptation to the data. The corresponding equation is:

$$g\left(u\right)=1-\max\left\{0,\left(1-\frac{p+1}{2}\left|u\right|\right)^{n}\right\} \tag{3}$$

where p and n are parameters.

Artificial neural network groups [35] are employed to combine several standard neural networks to yield approximators of piecewise continuous functions. The combined neural networks have standard activation functions, while the combination operators produce discontinuous outputs. The specific goal of this model is to estimate discontinuous functions. This is not our aim, since we intend to solve the same kind of problems as standard feedforward neural networks.

Piecewise linear activation functions have been considered in [1,19,20], but they have important shortcomings. First of all, they are not constrained to be monotonic. This means that their output might be similar for most of the input patterns, which would imply that the neuron does not make any useful calculation. Moreover, the functions in [1] must behave like the identity function for very large or very small inputs, so that their output is not bounded. These facts imply that the classic result of [17] for monotonic, bounded activation functions does not apply to them, so that their universality must be investigated. Nevertheless, it must be noted that for specific non-monotonic activation functions, universality can be proved, as seen in [7].

In fact, the idea of a monotonic piecewise polynomial activation function is not new. The Vapnik–Chervonenkis (VC) dimension of such neural networks for binary classification problems was studied in [3,27]. These theoretical works were aimed to estimate the complexity of the classification problems that such networks could learn. Their motivation was that it was easier to formally study the VC dimension of piecewise polynomial activation functions than standard sigmoid functions. Hence, there was no intention to adapt piecewise polynomial functions to the data, but to use them as approximations to the standard sigmoid functions.

Recent theoretical and empirical work in statistical machine learning has demonstrated the importance of learning algorithms for deep architectures. Moreover, the design of activation functions that enable fast training of accurate deep neural networks is an active area of research. Promising results of the influence of rectified linear units [13,23] have been presented in comparison with logistic sigmoid activations on image classification tasks. Also, Goodfellow et al. [14] introduced the *maxout* activation function, which can approximate any convex function of the input. Springenberg and Riedmiller [28] extended the previous work by replacing the max function with a probabilistic max function, and Gulcehre et al. [15] provided an activation function with an L_p norm. In all these cases, the proposed activation functions have no tunable parameters, so these approaches are far from our ideas. An approach which is much closer to our purposes is given in [7]. The activation function is learned along with the likelihood of the unit being relevant to the current test pattern. In their case, a mix of supervised and unsupervised learning is employed, while our approach employs supervised learning only.

Some variable activation functions for extreme learning machines (ELM) have been proposed, although they are parametric approaches since the number of tunable parameters is only two or three: three parameters (shift, position and mapping factor) in [33], which uses particle swarm optimization to adjust them, and two parameters for the fuzzy activation function in [18], which features compatible performance with the classical sigmoid function, while it can be implemented in a simple way in hardware devices with reduced computational capabilities.

3 Methodology

In this section, our proposal is presented. First, we specify the architecture of the network (Sect. 3.1). Then the learning procedures for the connection weights and the activation function are detailed in Sect. 3.2. The termination condition is specified in Sect. 3.3. After that, the training process is summarized in Algorithm 1. Finally, its universal approximation property is formally established (Sect. 3.4).

3.1 Architecture

The activation function of a neuron in a feedforward neural network is a nonlinear function g of the synaptic potential $u \in \mathbb{R}$. The output of a neural network with N inputs and L hidden units is given by:

$$y = \sum_{i=1}^{L} w_{2,i} g_i (u_i) \tag{4}$$

$$u_i = \sum_{j=1}^{N} w_{1,i,j} x_j \tag{5}$$

where $x_j \in \mathbb{R}$ is the jth input to the network, $w_{1,i,j} \in \mathbb{R}$ is the connection weight which connects the jth input to the ith hidden neuron, g_i is the activation function of the ith hidden neuron, u_i is the argument of the activation function of the ith hidden neuron, and $w_{2,i} \in \mathbb{R}$ is the connection weight which connects the ith hidden neuron to the output. The bias parameters are integrated into the connection weight set by assuming that the last input is held constant, i.e., $x_N = -1$. Please note that here networks with a single real output are considered, although functions with more outputs can be approximated by using several independent networks.

In this work, we propose to use monotonic piecewise polynomial activation functions for feedforward neural networks. Each member of this class of functions has an associated set of nodes, i.e., the points of the function domain where it switches from one polynomial to another. Let m be the number of nodes, which must be finite, so that the set of nodes is given by:

$$Q = \{q_1, q_2, \ldots, q_m\} \tag{6}$$

so that the nodes are sorted in increasing order, and there are no repeated nodes:

$$q_1 < q_2 < \cdots < q_m \tag{7}$$

In what follows, the number of nodes m is assumed to be a constant to be fixed before training and it will also be assumed that all neurons have the same number of nodes. The nodes should concentrate on those regions of the real line where synaptic potentials occur. This way, the nodes and their associated trainable parameters will not be wasted on regions of the real line where nothing happens during the operation of the neural network. With this in mind, we propose that the nodes aim to approximate the m-quantiles of the synaptic potential values u:

$$P\left(u \leq q_k\right) \approx \frac{k}{m} \tag{8}$$

where P stands for probability. Please note that (8) does not prescribe a equidistant distribution of the nodes on the real line. It means that the probability that a synaptic potential falls into one of the intervals between consecutive nodes is approximately the same for all these intervals, while the sizes of the intervals themselves could be largely different. This way, the density of the nodes on the real line will approximately match the probability density of the synaptic potentials. Hence, the nodes are placed where the synaptic potentials occur during the operation of the neural network. Under these conditions, the activation function approximates the cumulative distribution function of the synaptic potential:

$$g\left(\hat{u}\right) \approx P\left(u \leq \hat{u}\right) \tag{9}$$

Moreover, we aim to approximate the support of the distribution of u with the interval $[q_1, q_m]$, so that:

$$P\left(u \notin [q_1, q_m]\right) \approx 0 \tag{10}$$

Please note that (8) and (10) are compatible, provided that $P(u = q_1)$ is approximately $\frac{1}{m}$, since the interval $[q_1, q_m]$ in (10) is closed. In any case, both equations are approximations.

With (8) and (9) in mind, we define:

$$g\left(q_k\right) = \frac{k}{m} \tag{11}$$

It must be noted that (11) does not cause loss of generality in the set of functions that g can approximate, because the number of nodes m can be increased as required by the steepness of the function to be approximated. In this work, piecewise polynomials are used for g. Mixtures of logistics could also be employed, but we proposed piecewise polynomials because they are faster to evaluate in FPGAs and microcontrollers, which are extremely slow in computing transcendental functions. Thereby, our choice reduces the computational complexity in both the training and test phases for these hardwares. Also, it is interesting to note that (11) does not prescribe that the nodes q_k are equally spaced, but that the probability that the synaptic potential u falls between any inter-node interval $[q_1, q_m]$ is approximately the same. The rationale behind approximating the distribution function is that the distribution of the neuron output becomes approximately uniform on the output range, so that the amount of information carried by the neuron output is maximized.

The specific form of g depends on the order of the polynomials to be used. For zeroth-order polynomials, a step function is considered:

$$g_0(u) = \begin{cases} 0 & \text{if } u < q_1 \\ \frac{k}{m} & \text{if } q_k \le u < q_{k+1} \\ 1 & \text{if } u \ge q_m \end{cases} \tag{12}$$

where $k \in \{1, \dots, m-1\}$. For first-order polynomials, a piecewise linear function is considered:

$$g_1(u) = \begin{cases} 0 & \text{if } u < q_1 \\ \frac{k}{m} + \frac{u - q_k}{m(q_{k+1} - q_k)} & \text{if } q_k \le u < q_{k+1} \\ 1 & \text{if } u \ge q_m \end{cases} \tag{13}$$

where again $k \in \{1, \dots, m-1\}$.

It is also possible to employ piecewise third-degree interpolating polynomials [12], so that the resulting activation function is monotonic and continuous, and its first derivative is also continuous. Piecewise quadratic polynomials are not considered because they are more complex to find and evaluate than linear ones, and they have discontinuous first derivatives unlike cubic ones, so they are not advantageous.

The expression for the third-degree case $g_3(u)$ varies depending on the particular interpolating piecewise cubic polynomial which is chosen. In particular, if natural splines are chosen, i.e., the second derivatives are null at $u = q_1$ and $u = q_m$, then it reads as follows:

$$g_3(u) = \begin{cases} 0 & \text{if } u < q_1 \\ S_k(u) & \text{if } q_k \le u < q_{k+1} \\ 1 & \text{if } u \ge q_m \end{cases} \tag{14}$$

where

$$\begin{aligned} S_k(u) = {} & \frac{t_{k+1}}{6h_k}(u - q_k)^3 + \frac{t_k}{6h_k}(q_{k+1} - u)^3 \\ & + \left(\frac{k+1}{mh_k} - \frac{t_{k+1}h_k}{6}\right)(u - q_k) + \left(\frac{k}{mh_k} - \frac{t_k h_k}{6}\right)(q_{k+1} - u) \end{aligned} \tag{15}$$

$$h_k = q_{k+1} - q_k \tag{16}$$

$$b_k = \frac{1}{mh_k} \tag{17}$$

$$v_k = 2(h_{k-1} + h_k) \tag{18}$$

$$d_k = 6(b_k - b_{k-1}) \tag{19}$$

$$t_k = g_3''(q_k) \tag{20}$$

$$t_0 = t_m = 0 \tag{21}$$

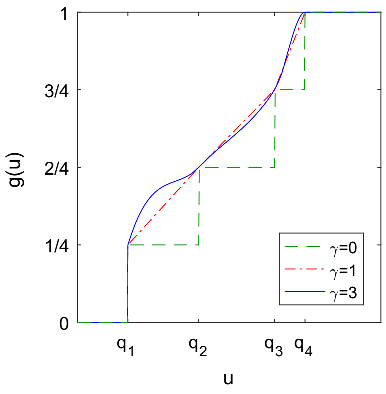


Fig. 1 Example of activation functions for different values of the polynomial order γ. The number of nodes is $m = 4$

$$\begin{pmatrix} v_1 & h_1 & & & & \\ h_1 & v_2 & h_2 & & & \\ & h_2 & v_3 & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & h_{m-2} \\ & & & & h_{m-2} & v_{m-1} \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_{m-2} \\ t_{m-1} \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{m-2} \\ d_{m-1} \end{pmatrix} \tag{22}$$

where (22) is a tridiagonal linear system of equations with $m - 1$ unknowns and $m - 1$ equations.

In what follows, we note the order of the used polynomials as γ, i.e., $\gamma = 0$ for zeroth-order polynomials, $\gamma = 1$ for first-order polynomials, and $\gamma = 3$ for third-order polynomials. Figure 1 depicts an example for $m = 4$ of the activation functions corresponding to these values of γ. As seen, all functions exhibit an abrupt change at $u = q_1$. This is because we aim to approximate the support of the distribution function of the synaptic potential u within the interval $[q_1, q_m]$ (Eq. 10). Therefore, the distribution function is assumed to be approximately zero for all $u < q_1$. The jump is unavoidable, but it occurs at an extreme value of the synaptic potential. Moreover, the activation function learning algorithm (Sect. 3.2) reduces q_1 until the case $u < q_1$ is unlikely. For $\gamma = 3$, we need to set $g\,(q_1 - \epsilon) = 0$ for some small $\epsilon > 0$.

In order to use the gradient method to learn the connection weights (Sect. 3.2), the derivative of the activation function is normally employed. For zeroth- and first-order polynomials, this poses difficulties, so alternative solutions are required. In the case of first-order polynomials, the derivative of g_1 is not defined at the nodes. We propose to use the right derivative of g_1, which is defined for all real numbers although it is not continuous:

$$g_1'\,(u) = \begin{cases} 0 & \text{if } u < q_1 \\ \frac{1}{m(q_{k+1}-q_k)} & \text{if } q_k \leq u < q_{k+1} \\ 0 & \text{if } u \geq q_m \end{cases} \tag{23}$$

In the case of zeroth-order polynomials the derivative of g_0 is always zero, which is not usable for the gradient method. In order to overcome this difficulty, we propose to use the same function as in the case of first-order polynomials:

$$g_0'\,(u) = g_1'\,(u) \tag{24}$$

The rationale behind (24) is that, since g is meant to approximate the distribution function of the synaptic potential, its derivative should get closer to the probability density function of the synaptic potential as the number of samples of the synaptic potential and m grow, no matter the specific form of g. Therefore, g_0' is regarded as a quickly evaluable approximation to the probability density function of the synaptic potential.

For third (and higher)-order polynomials, the derivative does not pose any difficulty since it is continuous in $\mathbb{R}$, so we use it as g_3'. It reads as follows:

$$g_3'(u) = \begin{cases} 0 & \text{if } u < q_1 \\ S_k'(u) & \text{if } q_k \leq u < q_{k+1} \\ 1 & \text{if } u \geq q_m \end{cases} \tag{25}$$

$$S_k'(u) = \frac{t_{k+1}}{2h_k}(u - q_k)^2 - \frac{t_k}{2h_k}(q_{k+1} - u)^2 + b_k - \frac{h_k}{6}(t_{k+1} - t_k) \tag{26}$$

3.2 Learning Procedure

The learning rules for the connection weights and the activation function are derived next.

First, the connection weight learning is considered. Let us consider the following training set with M training patterns:

$$\mathcal{T} = \{(\mathbf{x}_1, z_1), \ldots, (\mathbf{x}_M, z_M)\} \tag{27}$$

where $\mathbf{x}_r \in \mathbb{R}^N$ is the rth input and $z_r \in \mathbb{R}$ is the rth desired output. For batch learning, the goal is to minimize the mean squared error (MSE):

$$\text{MSE} = \frac{1}{M}\sum_{r=1}^{M} E_r \tag{28}$$

$$E_r = (y_r - z_r)^2 = \left(\left(\sum_{i=1}^{L} w_{2,i}\, g_i\left(\sum_{j=1}^{N} w_{1,i,j} x_{r,j}\right)\right) - z_r\right)^2 \tag{29}$$

where $y_r \in \mathbb{R}$ is the rth actual output of the network. We use the stochastic gradient method in order to obtain an online learning rule. This involves the computation of the partial derivatives of the squared error E_r associated with the current training pattern $(\mathbf{x}_r, z_r)$, with respect to the synaptic weights. Consequently, the online learning rules for the connection weights are:

$$\bar{w}_{1,i,j} = w_{1,i,j} - \eta_1 \frac{\partial E_r}{\partial w_{1,i,j}} \tag{30}$$

$$\bar{w}_{2,i} = w_{2,i} - \eta_2 \frac{\partial E_r}{\partial w_{2,i}} \tag{31}$$

where η_1 and η_2 are the learning rates for the hidden layer and the output neuron, respectively, and the bars indicate the updated weights. Please note that two different learning rates η_1 and η_2 are defined because the structure of the hidden layer is different from that of the output layer, since polynomial activation functions are used in the hidden layer, while the output layer uses a pure linear activation function.

For the output neuron, the computation of the partial derivatives is as follows:

$$\frac{\partial E_r}{\partial w_{2,i}} = 2\left(y_r - z_r\right) g_i\left(\sum_{j=1}^{N} w_{1,i,j} x_{r,j}\right) \tag{32}$$

On the other hand, for the hidden layer the equation varies depending on the order of the polynomials and the intervals of the activation function. From (23) and (24), we have:

$$\frac{\partial E_r}{\partial w_{1,i,j}} = 2\frac{\left(y_r - z_r\right) w_{2,i} x_{r,j}}{m\left(q_{i,k+1} - q_{i,k}\right)}, \text{ if } \gamma \in \{0, 1\} \wedge u_{r,i} \in \left[q_{i,k}, q_{i,k+1}\right) \tag{33}$$

$$\frac{\partial E_r}{\partial w_{1,i,j}} = 2\left(y_r - z_r\right) w_{2,i} g'_{3,i}\left(u_{r,i}\right) x_{r,j}, \quad \text{if } \gamma = 3 \tag{34}$$

where $u_{r,i}$ stands for the synaptic potential at the ith neuron for the rth training pattern and $q_{i,k}$ stands for the kth node of the ith neuron. Please note that $g'_{3,i}(u)$ is given by (25).

Next, the adaptation of the activation function to the training data is described. Two learning parameters $\lambda, \beta \in (0, 1]$ are considered. Let i be the index of the hidden neuron whose activation function g_i is to be updated, with $\bar{g}_i$ noting the updated version. As done in Sect. 3.2, an online learning procedure is considered, and the current training pattern is noted $(\mathbf{x}_r, z_r)$.

The overall goal of the learning procedure for the nodes is that, for each incoming training pattern $(\mathbf{x}_r, z_r)$ and all neurons i, one node of each neuron is slightly displaced toward the current value of the synaptic potential $u_{r,i}$. This way, as more patterns are presented to the network, nodes are placed in the regions of the real line where more synaptic potential values occur. The λ parameter controls how often the nodes of the activation function are updated: The larger the λ, the more often they are updated. On the other hand, the β parameter controls the relative importance of the current argument of the activation function $u_{r,i}$ in the node update: The larger the β, the more the importance which is given to $u_{r,i}$. If $u_{r,i} \in \left[q_{i,k}, q_{i,k+1}\right)$ with $k \in \{1, \ldots, m-1\}$, then the nodes of the ith hidden neuron are updated as follows:

– If $w_{2,i}\left(y_r - z_r\right) > 0$ and $\left|\frac{2w_{2,i}}{m}\right| < \lambda\left|y_r - z_r\right|$, then the node update is:

$$\bar{q}_{i,k} = \beta u_{r,i} + (1-\beta)\, q_{i,k+1} \tag{35}$$

– If $w_{2,i}\left(y_r - z_r\right) < 0$ and $\left|\frac{2w_{2,i}}{m}\right| < \lambda\left|y_r - z_r\right|$, then the node update is:

$$\bar{q}_{i,k+1} = \beta u_{r,i} + (1-\beta)\, q_{i,k} \tag{36}$$

– If $w_{2,i}\left(y_r - z_r\right) = 0$ or $\left|\frac{2w_{2,i}}{m}\right| \geq \lambda\left|y_r - z_r\right|$, then no node update is performed.

Please note that (35) and (36) preserve the order of the nodes because $u_{r,i} \in \left[q_{i,k}, q_{i,k+1}\right)$ and $\beta \in (0, 1]$.

The β parameter plays the role of a learning rate for the nodes of the activation function. Therefore, its interpretation and management is similar to that of the learning rates for the weight vectors η_1 and η_2. In other words, β might be fixed along with η_1 and η_2. They must not be set too high so that the decrease in the training error is steady, and they must not be too low so that the decay of the training error is fast enough. The λ parameter controls how much training error a training sample must exhibit in order to trigger the node learning procedure. Therefore, it decouples the learning of the weights, which is done for all training patterns, from the learning of the nodes, which is done at a slower rate. It must be adjusted in a similar way to the learning rates, i.e., it must be high enough that the decrease in the training error is not too slow, but it must be small enough to allow a steady decrease in the error.

We still have to manage extreme values of the argument of the activation function, i.e., $u_{r,i} < q_{i,1}$ or $u_{r,i} > q_{i,m}$. According to (10), all observed values of $u_{r,i}$ should lie in the interval $[q_1, q_m]$. This is readily accomplished by these additional update rules:

– If $u_{r,i} < q_{i,1}$, then the node update is as follows:

$$\bar{q}_{i,1} = u_{r,i} - \kappa \tag{37}$$

– If $u_{r,i} > q_{i,m}$, then the node update is as follows:

$$\bar{q}_{i,m} = u_{r,i} + \kappa \tag{38}$$

where $\kappa > 0$ is a small constant.

These additional update rules overcome the problem of the "vanishing gradient" which is common to many saturated activation functions such as the rectified linear units (ReLU, [21]). When the argument of the activation function $u_{r,i}$ of a neuron i is higher than the largest node $q_{i,m}$, then $q_{i,m}$ is increased. Conversely, when $u_{r,i}$ is smaller than the lowest node $q_{i,1}$ then $q_{i,1}$ is decreased. That is, the non-saturated range of the activation function is enlarged as necessary. That way, the argument of the activation function $u_{r,i}$ is kept within the non-saturated range $[q_1, q_m]$ of the activation function.

It must be highlighted that this procedure for extreme values of $u_{r,i}$ is not guaranteed to decrease the error, i.e., it is not covered by Theorem 1 (see Sect. 3.4). Nevertheless, this procedure will only be activated when $u_{r,i} \notin \left[q_{i,1}, q_{i,m}\right]$, i.e., when the current value of the synaptic potential is outside the range of previously observed synaptic potential values. Therefore, the probability that the procedure is activated will decrease as learning progresses, since $\left[q_{i,1}, q_{i,m}\right]$ can only grow.

For each hidden neuron m values are drawn independently from the uniform distribution on the interval $[-c, c]$ and then sorted in increasing order to yield the initial values of the nodes $q_1, \ldots, q_m$, where c is a parameter of the algorithm. This way, it is ensured that (7) holds at the beginning of the learning process. It is worth noting that all the node update procedures described in Sect. 3.2 preserve the relative order of the nodes. Therefore, condition (7) holds at all times.

3.3 The Termination Criterion

Overfitting is a well-known problem of predictive algorithms. It is caused by an over specialization of the training procedure on the training set of patterns [16]. As usual in feedforward neural networks, a straightforward alternative is to split the data set in training, validation and test sets, in order to use the validation set to control the overfitting effects through a termination criterion based on the performance on the validation set.

The trend of the mean squared error (MSE) is difficult to estimate since it fluctuates in each epoch. In order to alleviate this effect, window functions have been designed to be used as filters for the MSE values. There are several window functions that could be applied. Two of the most used ones are as follows:

1. Rectangular window

$$w(n) = 1 \tag{39}$$

2. Hamming window

$$w(n) = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi n}{N-1}\right) \tag{40}$$

where N represents the width, in samples, of a discrete-time, geometric window function $w(n), 0 \leq n \leq N-1$. The window is divided by the energy of the function in order to reduce the total energy to 1.

We have selected the Hamming window with $N = 15$ to use as the filter for the validation MSE. The termination criterion is to halt the learning process as the Hamming window smoothed function ascending for a number of epochs equal to the selected N.

Algorithm 1 PPAFNN Training algorithm

```
1:  Initialization of hidden neurons and connection weights
2:  repeat
3:      Shuffle the training samples in random order.
4:      for all Training patterns do
5:          Fetch a new training pattern $(\mathbf{x}_r, z_r)$.
6:          Compute the gradient of the error by (32), (33) and (34)
7:          Update the connection weights with (30) and (31)
8:          Update the hidden neurons according to (35), (36), (37) and (38)
9:      end for
10: until termination criterion described in section 3.3
```

3.4 Theoretical Properties

At this point, it is important to ensure that the neural network model defined in Sect. 3.1 has the capability to approximate any Borel measurable function with any degree of accuracy. Following [17], it is said that g is a squashing function if it is non-decreasing, it has at most countably many discontinuities, and the two following conditions hold:

$$\lim_{u \to \infty} g(u) = 1 \tag{41}$$

$$\lim_{u \to -\infty} g(u) = 0 \tag{42}$$

The definitions in Sect. 3.1 imply that all the activation functions g_γ considered in this paper are squashing functions. Then from [17] it follows that our proposed PPAFNN model has the universal approximation property, i.e., PPAFNN networks can approximate any continuous function uniformly on any compact set and any measurable function arbitrarily well.

The proposed update procedure is justified by the following formal results.

Proposition 1 *The application of Eq.* (35) *under the condition:*

$$\left(w_{2,i}\left(y_r - z_r\right) > 0\right) \wedge \left|\frac{2w_{2,i}}{m}\right| < \lambda \left|y_r - z_r\right| \tag{43}$$

ensures that the squared error E_r decreases or remains the same.

Proof From the update procedure, we know that $u_{r,i} \in \left[q_{i,k}, q_{i,k+1}\right)$. Therefore,

$$g_i\left(u_{r,i}\right) = \frac{k + \delta_{i,k}\left(u_{r,i}\right)}{m} \tag{44}$$

where $\delta_{i,k}\left(u_{r,i}\right) \in [0,1]$. The exact form of $\delta\left(u_{r,i}\right)$ depends on the order of the polynomials γ.

If $w_{2,i}\left(y_r - z_r\right) > 0$ and $\left|\frac{2w_{2,i}}{m}\right| < \lambda\left|y_r - z_r\right|$, then the update Eq. (35) implies that:

$$u_{r,i} \leq \bar{q}_{i,k} \leq q_{i,k+1} \tag{45}$$

Therefore,

$$\bar{g}_i\left(u_{r,i}\right) = \frac{k - 1 + \bar{\delta}_{i,k-1}\left(u_{r,i}\right)}{m} \tag{46}$$

where the bars correspond to the values obtained after executing the update. Moreover, from (4):

$$\begin{aligned} y_r - \bar{y}_r &= \sum_{s=1}^{L} w_{2,s} g_s\left(u_{r,s}\right) - \sum_{s=1}^{L} w_{2,s} \bar{g}_s\left(u_{r,s}\right) \\ &= w_{2,i} g_i\left(u_{r,i}\right) - w_{2,i} \bar{g}_i\left(u_{r,i}\right) \end{aligned} \tag{47}$$

Then from (44), (46) and (47):

$$\begin{aligned} y_r - \bar{y}_r &= w_{2,i}\left(\frac{k + \delta_{i,k}\left(u_{r,i}\right)}{m} - \frac{k - 1 + \bar{\delta}_{i,k-1}\left(u_{r,i}\right)}{m}\right) \\ &= w_{2,i}\frac{\delta_{i,k}\left(u_{r,i}\right) - \bar{\delta}_{i,k-1}\left(u_{r,i}\right) + 1}{m} \end{aligned} \tag{48}$$

Since $w_{2,i}\left(y_r - z_r\right) > 0$, there are two possible cases: (a) $\left(w_{2,i} > 0\right) \wedge \left(y_r - z_r > 0\right)$; (b) $\left(w_{2,i} < 0\right) \wedge \left(y_r - z_r < 0\right)$.

For case (a), since $\delta_{i,k}\left(u_{r,i}\right), \bar{\delta}_{i,k-1}\left(u_{r,i}\right) \in [0,1]$, from (48) we obtain:

$$0 \leq y_r - \bar{y}_r \leq \frac{2w_{2,i}}{m} \tag{49}$$

On the other hand, since $y_r - z_r > 0$, $w_{2,i} > 0$, $\lambda \in (0,1]$ and $\left|\frac{2w_{2,i}}{m}\right| < \lambda\left|y_r - z_r\right|$ we have:

$$\frac{2w_{2,i}}{m} \leq y_r - z_r \tag{50}$$

From (49) and (50):

$$0 \leq y_r - \bar{y}_r \leq y_r - z_r \tag{51}$$

$$-y_r \leq -\bar{y}_r \leq -z_r \tag{52}$$

$$z_r \leq \bar{y}_r \leq y_r \tag{53}$$

$$\bar{E}_r \leq E_r \tag{54}$$

That is, the new squared error $\bar{E}_r$ is lower than or equal to the old squared error E_r, as required.

For case (b), since $\delta_{i,k}\left(u_{r,i}\right), \bar{\delta}_{i,k-1}\left(u_{r,i}\right) \in [0,1]$, from (48) we obtain:

$$\frac{2w_{2,i}}{m} \leq y_r - \bar{y}_r \leq 0 \tag{55}$$

On the other hand, since $y_r - z_r < 0$, $w_{2,i} < 0$, $\lambda \in (0,1]$ and $\left|\frac{2w_{2,i}}{m}\right| < \lambda\left|y_r - z_r\right|$ we have:

$$y_r - z_r \leq \frac{2w_{2,i}}{m} \tag{56}$$

From (55) and (56):

$$y_r - z_r \leq y_r - \bar{y}_r \leq 0 \tag{57}$$

$$-z_r \leq -\bar{y}_r \leq -y_r \tag{58}$$

$$y_r \leq \bar{y}_r \leq z_r \tag{59}$$

$$\bar{E}_r \leq E_r \tag{60}$$

And again the new squared error $\bar{E}_r$ is lower than or equal to the old squared error E_r, as required. □

Proposition 2 *The application of Eq.* (36) *under the condition:*

$$\left(w_{2,i}\left(y_r - z_r\right) < 0\right) \wedge \left|\frac{2w_{2,i}}{m}\right| < \lambda \left|y_r - z_r\right| \tag{61}$$

ensures that the squared error E_r decreases or remains the same.

Proof See "Appendix." □

Theorem 1 *If $u_{r,i} \in [q_1, q_m]$ then the above-described activation function update procedure ensures that the squared error E_r decreases or remains the same.*

Proof The result can be inferred from Propositions 1 and 2. □

The formal results proved in this subsection ensure that our proposal is able to approximate any Borel measurable function with any degree of accuracy, so that its scope of application matches that of multilayer perceptrons with standard activation functions. Furthermore, it has also been proved that the training error decreases whenever the activation function update is used for $u_{r,i} \in [q_1, q_m]$. This means that the activation function update effectively contributes to the adaptation of the neural network to the training set. The computational load with respect to standard multilayer perceptrons depends on two factors. The most important one is the degree of the piecewise polynomials γ. For $\gamma = 0$ or $\gamma = 1$ the computational complexity of the activation function is small, since the expressions that must be evaluated (12) or (13) are pretty simple. However, for $\gamma > 1$ the complexity rises significantly (14). Therefore, if the flexibility of the activation function must be increased, it is likely that increasing the number of nodes m with $\gamma = 0$ or $\gamma = 1$ yields better results than setting $\gamma = 3$.

4 Experimental Results

In this section, the computational experiments are described and their results are reported.

Several test cases were analyzed to verify the presented models, comparing their results with the most used supervised learning algorithm to train feedforward neural models, namely the backpropagation algorithm [25] with different activation functions. The activation function used to compare the proposed algorithms are as follows: the sigmoid function (Sig) [25], the trainable amplitude of the sigmoid function (Amp) [30], the leaky rectified linear unit (ReLU) [21] and the function shape autotuning (Shape) [8].

The tests have been carried out with small architectures in order to prove the proposed activation functions with non-complex neural models. In this way, the capacity of the proposed functions in this type of architecture has been checked and we will be able to face more complex systems in future works.

These tests were carried out using a tenfold cross-validation for the training and generalization sets, respectively, including a validation set to decide the termination criterion (20% of the training set). All data sets have been normalized between 0 and 1 in order to carry out a correct implementation of the method, where the normalized values (V_{nor}) are calculated with the next equation: $V_{\text{nor}} = (\text{Value} - V_{\min})/(V_{\max} - V_{\min})$, being Value the value of the data set, $V_{\min}$ the minimum value of the data set, and $V_{\max}$ the maximum value of the data set.

The configuration data set for the proposed models is as follows: initial quantile coefficient parameter $c = 1$, number of nodes $m = 20$, connection weight learning rates $\eta_1 = \eta_2 = 0.01$, activation function learning parameters $\beta = 0.7$, $\lambda = 1$, and $\kappa = 0.001$. For the competing backpropagation algorithm the learning rate (η) value was fixed to 0.2. These values have been determined manually, and then they have been found to work correctly for a wide range of data sets, as we will see. Please see Sect. 3.2 for explanations of the effects of the values of each parameter on the learning process.

Figure 2 shows the mean squared error (MSE) evolution (left side axis) for the training, validation and Hamming window processes and the accuracy evolution (right side axis) according to the number of epochs of a random tenfold cross-validation execution for the proposed models: Constant model (a), Linear model (b), Cubic model (c), while Fig. 3 shows the same for the traditional models: sigmoid activation function (a), the trainable amplitude of the sigmoid function (b), the leaky rectified linear unit (c) and the function shape autotuning (d) using data from the well-known Diabetes set. The employed neural architecture contained five neurons in a single hidden layer.

In all seven graphs, the vertical line indicates the average epoch when each algorithm fulfills its termination criterion. Furthermore, the graphs show the accuracy value at these epochs. Also, the horizontal line indicates the maximum value of the accuracy for each algorithm (higher is better).

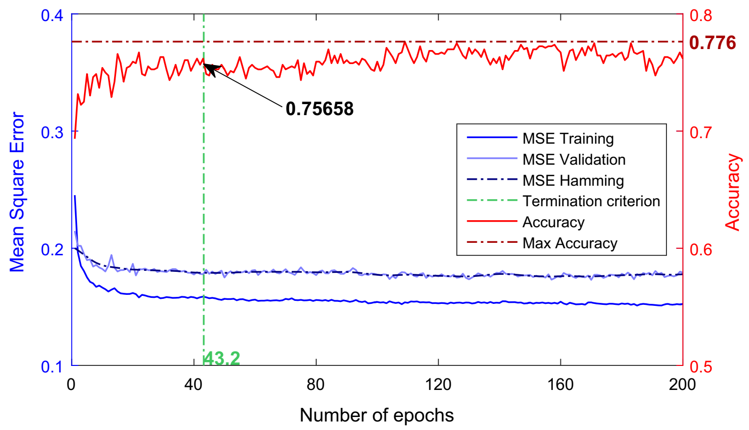
Also, Table 1 shows the accuracy of generalization ability in terms of mean and standard deviation obtained from the cases shown in Figs. 2 and 3. In the first column, the analyzed algorithms are shown, while the second column shows the mean and standard deviation with the notation mean $\pm$ standard deviation. From these data, it can be said that our approaches attain a better accuracy than their competitors for this experiment.

Figure 4 shows the average of the accuracy (a) and the termination criterion (b) of 100 runs, changing in each new run the initial values of the connection weights of the networks, of the tenfold cross-validation as a function of the number of neurons present in the hidden layer of the neural architecture for the proposed algorithms (Constant, Linear and Cubic) and the backpropagation algorithm with the different activation functions (Sig, Amp, ReLU and Shape). The Diabetes data set has been used for this purpose.

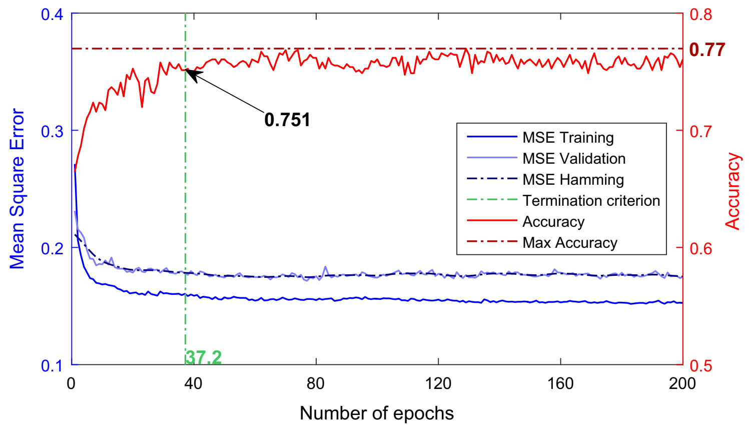
We have analyzed the performance of the proposed algorithms to test the generalization ability on a set of 11 real-world classification data sets obtained from the UCI machine learning repository [31], a set of 10 single-output Boolean functions from the MCNC benchmark [22] and a set of 12 real-world regression data sets obtained from the UCI machine learning repository [31].

The tests have been carried out with different neural architecture sizes (two, five and ten neurons in a single hidden layer) in order to check the performance of the proposed algorithms. Also, the tests have been performed by 100 runs of tenfold cross-validation in order to compute the mean and the standard deviation over the runs.

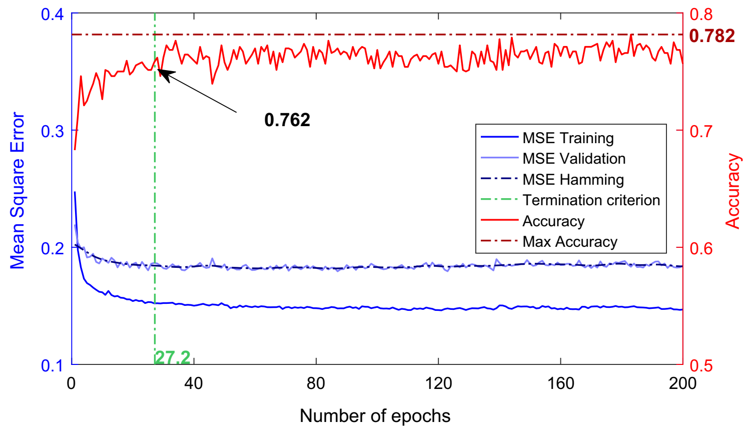
Table 2 shows the average of the classification performance calculated from Table 6, in which the generalization ability for eleven real data sets are reported according to the size of the neural architecture. Table 3 shows the average of the classification performance



(a) Constant



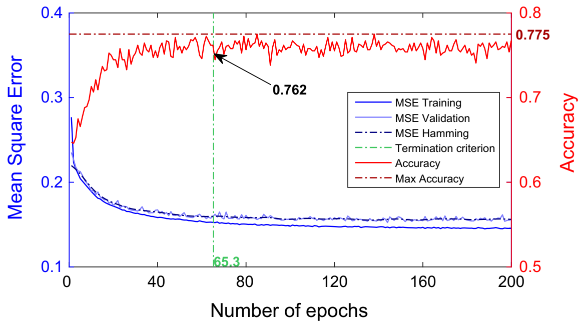
(b) Linear



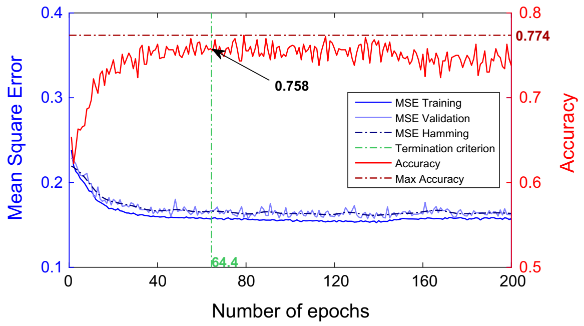
(c) Cubic

Fig. 2 Mean squared error (MSE) evolution for the training, validation and Hamming window processes and the accuracy evolution according to the number of epochs for the proposed models

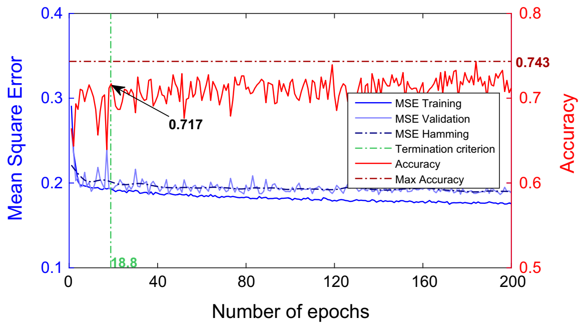
Fig. 3 Mean squared error (MSE) evolution for the training, validation and Hamming window processes and the accuracy evolution according to the number of epochs for the traditional models



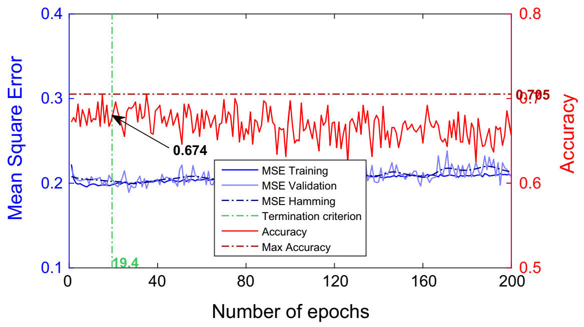
(a) Sig



(b) Amp



(c) ReLU



(d) Shape

Table 1 Generalization ability of the proposed (Constant and Linear) and the compared (Sig, Amp, ReLU and Shape) methods in terms of mean and standard deviation computed from Figs. 2 and 3

Activation function	Accuracy
Constant	75.88% ± 1.05
Linear	75.34% ± 1.46
Cubic	75.92% ± 1.22
Sig	75.29% ± 2.10
Amp	74.60% ± 2.25
ReLU	70.95% ± 1.51
Shape	66.90% ± 1.63

calculated for the Boolean functions from Table 7. Finally, Table 8 shows the regression ability for twelve real data sets according to the size of the neural architecture, and Table 4 reports the average of the regression performance calculated from Table 8.

As seen in the reported results, our approaches (Constant, Linear and Cubic) yield consistently better results in the vast majority of the cases, for both classification and regression applications.

We have analyzed the evolution of generalization ability of the eleven data sets according to the number of neurons in the hidden layer. Figure 5 shows the average of the accuracy of the eleven data sets of the tenfold cross-validation as a function of the number of neurons present in the hidden layer of the neural architecture for the proposed algorithms (Constant, Linear and Cubic) and the backpropagation algorithm with the different activation functions (Sig, Amp, ReLU and Shape).

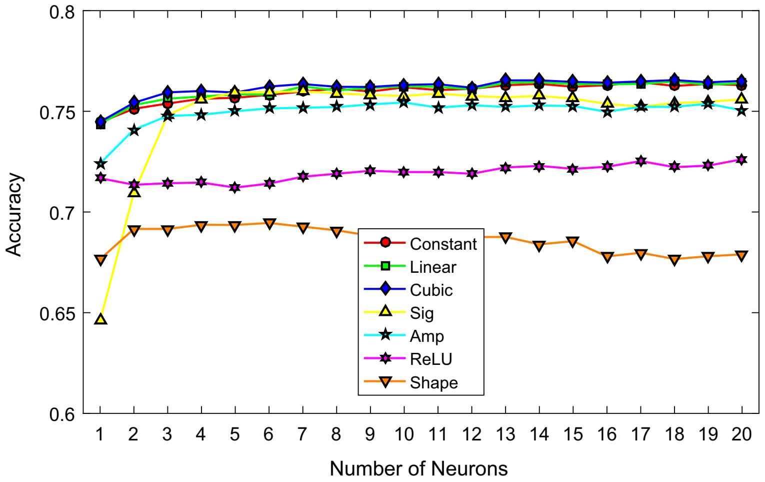
A statistical test has been carried out, the Welch's t test, in order to determine if there is a difference between the traditional and proposed methods. Welch's t test, or unequal variances t test, is a two-sample location test which is used to test the hypothesis that two populations have equal means in which the two samples can have unequal variances and unequal sample sizes. Welch's t test defines the statistic t by the following formula:

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \tag{62}$$

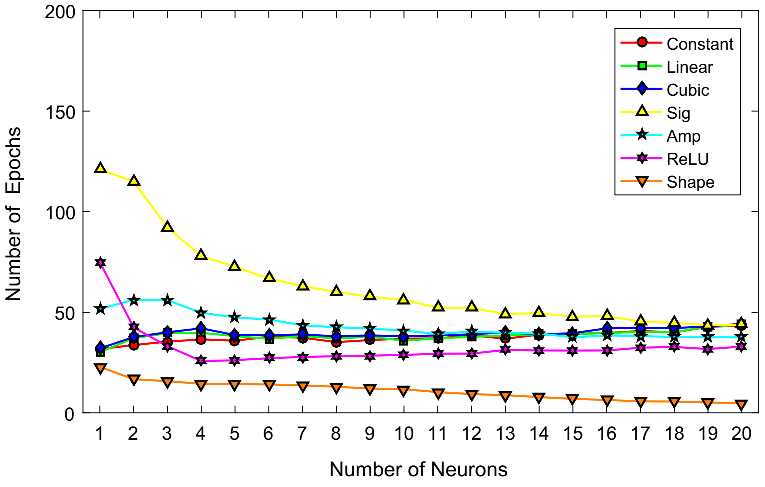
where $\overline{X_1}$, S_1^2 and N_1 are the first sample mean, sample variance and sample size, respectively. In the case where it is not assumed that the two data samples are from populations with equal variances, the test statistic under the null hypothesis has an approximate Student's t distribution with a number of degrees of freedom given by Satterthwaite's approximation. As Welch's t test performs a comparison between two samples, we have compared the Cubic methods and the Sig methods, since they are the best methods among the proposed and traditional ones, respectively.

Table 5 shows the p values for the null hypothesis that the data of the proposed (Cubic) and traditional (Sig) models come from independent random samples from normal distributions with equal means and unequal variances, depending on the number of neurons in the architecture for the real classification data sets. As seen, the p values are quite small, which means that the performance differences between the traditional and proposed methods are statistically significant.

In general terms, it can be said that our approaches quickly attain their best performance on the validation set, which is better than the best performance of the traditional approaches



(a) Accuracy



(b) Termination criterion

Fig. 4 Average of the accuracy (**a**) and the number of epochs until the termination criterion is met (**b**) of 100 runs of the proposed (Constant, Linear and Cubic) and the backpropagation algorithms with the different activation functions (Sig, Amp, ReLU and Shape) as a function of the number of neurons present in the hidden layer of the neural architecture

(Table 1). Moreover, the performance differences have been found to be of strong statistical significance (Table 5).

5 Discussion

Flexibility is one of the main advantages of our approach, when compared to standard activation functions. The learning of the activation function of each neuron from the training

Table 2 Generalization ability in terms of accuracy (Acc) and termination criterion (Ter) obtained with the proposed algorithms (Linear, Constant and Cubic) and with four different activation functions (Sig, Amp, ReLU and Shape), on a set of real-world problems from the UCI data set (see Table 6)

Transfer function	# Neurons			
	5		10	
	Acc	Ter.	Acc	Ter.
Constant	84.52 ± 1.47	46.0 ± 10.3	**85.33** ± 1.54	45.1 ± 9.8
Linear	**84.72** ± 1.48	46.4 ± 11.5	84.99 ± 1.34	46.9 ± 10.1
Cubic	84.55 ± 1.25	43.7 ± 10.0	85.12 ± **1.17**	48.0 ± 11.5
Sig	83.82 ± 1.25	66.6 ± 20.4	84.11 ± 1.38	54.6 ± 15.4
Amp	82.73 ± **1.19**	40.8 ± 11.7	83.66 ± 1.33	35.7 ± 10.2
ReLU	81.59 ± 1.88	52.3 ± 13.4	79.46 ± 2.43	42.0 ± 12.6
Shape	81.34 ± 1.95	**20.2** ± **6.5**	79.77 ± 2.18	**16.6** ± **5.6**

Best results are shown in bold. See the main text for more details

Table 3 Generalization ability in terms of accuracy (Acc) and termination criterion (Ter) obtained with the proposed algorithms (Linear, Constant and Cubic) and with four different activation functions (Sig, Amp, ReLU and Shape), on a set of Boolean functions from the MCNC benchmark (see Table 7)

Transfer function	# Neurons			
	5		10	
	Acc	Ter.	Acc	Ter.
Constant	76.63 ± 2.85	65.8 ± 13.1	81.91 ± 3.35	77.0 ± 15.4
Linear	75.72 ± **2.49**	65.9 ± 14.1	81.05 ± 3.38	75.0 ± 12.5
Cubic	**79.20** ± 2.51	74.3 ± 14.8	**82.96** ± 3.11	84.0 ± 12.9
Sig	73.29 ± 3.14	106.8 ± 18.5	79.91 ± 3.77	128.6 ± 21.3
Amp	71.50 ± 3.51	79.5 ± 13.5	72.98 ± **2.49**	83.8 ± 13.2
ReLU	74.67 ± 3.18	65.7 ± 12.9	78.60 ± 2.85	71.0 ± 11.9
Shape	69.98 ± 2.72	**35.1** ± **11.6**	69.87 ± 3.93	**29.6** ± **11.5**

Best results are shown in bold. See the main text for more details

Table 4 Regression ability in terms of mean squared error (MSE) and termination criterion (Ter) obtained with the proposed algorithms (Linea, Constant and Cubic)) and with other four algorithms with different activation functions (Sig, Amp, ReLU and Shape), on a set of real-world problems from the UCI data set (see Table 8)

Transfer function	# Neurons			
	5		10	
	Acc	Ter.	Acc	Ter.
Constant	0.0098 ± 0.0010	71.4 ± 10.6	0.0093 ± **0.0009**	77.5 ± 10.5
Linear	**0.0093** ± 0.0009	71.9 ± 12.2	0.0090 ± **0.0009**	72.4 ± 10.6
Cubic	**0.0093** ± 0.0009	80.0 ± 11.5	**0.0088** ± 0.0009	72.4 ± 10.6
Sig	0.0097 ± **0.0007**	127.3 ± 19.0	0.0097 ± 0.0014	110.2 ± 20.0
Amp	0.0131 ± 0.0025	70.5 ± 13.9	0.0114 ± 0.0015	62.0 ± 15.2
ReLU	0.0141 ± 0.0009	52.6 ± 12.3	0.0131 ± 0.0011	57.3 ± 10.0
Shape	0.0122 ± 0.0014	**41.9** ± **9.0**	0.0126 ± 0.0020	**38.0** ± **8.6**

Best results are shown in bold. See the main text for more details

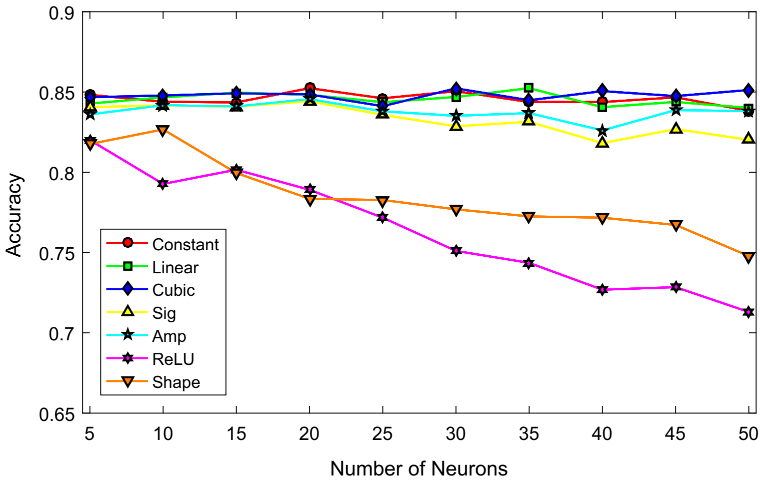


Fig. 5 Average of the accuracy of the eleven data sets of the tenfold cross-validation as a function of the number of neurons present in the hidden layer of the neural architecture for the proposed algorithms (Constant, Linear and Cubic) and the backpropagation algorithm with different activation functions (Sig, Amp, ReLU and Shape)

Table 5 p values for the null hypothesis that the data of the proposed and traditional models come from independent random samples from normal distributions with equal means and unequal variances, depending on the number of neurons in the architecture for the real classification data sets

Data sets	# Neurons		
	2 p values	5 p values	10 p values
Blood	1.39×10^{-6}	7.80×10^{-10}	7.80×10^{-10}
Cancer	7.76×10^{-9}	1.54×10^{-14}	3.26×10^{-9}
Card	6.81×10^{-4}	4.98×10^{-7}	1.55×10^{-9}
Climate	4.99×10^{-9}	6.29×10^{-7}	3.97×10^{-6}
Diabetes	7.05×10^{-9}	4.80×10^{-5}	9.29×10^{-6}
Fertility	5.33×10^{-9}	0.0034	0.6533
Heart	0.0083	0.0367	2.03×10^{-4}
Ionosphere	7.24×10^{-5}	0.1682	0.2625
Sonar	0.0070	0.1548	0.5612
Statlog	0.0123	0.2253	0.0138
Vertebral	0.0061	0.0023	0.5118

data ensures that our proposal has a greater adaptation capability than models with activation functions which are fixed or only allow for one or two tunable parameters (see Sect. 2). The model enjoys the universal approximation property (Sect. 3.4). Furthermore, its range of application includes regression and classification applications, as seen in Sect. 4. Flexibility must be controlled in order to avoid overfitting. For example, the number of nodes m in (11) must not be set too high. This can be managed by fixing m as a constant, or alternatively choosing it by cross-validation techniques. As mentioned before, our proposed learnable activation function is nonparametric in that the number of nodes m can be enlarged for bigger data set sizes. However, it is also interesting to note that it is not required that there is a node q_k for each training sample, so that each node can summarize a significant amount of training samples. This way, a balance between overfitting and underfitting can be attained, as demonstrated in the experiments.

The node update procedure for arguments of the activation function which lie in the range of previously observed values $\left[q_{i,1}, q_{i,m}\right]$ ensures that the error decreases, as proved by Theorem 1. For the rare cases such that extreme values outside of the previously observed range of values $\left[q_{i,1}, q_{i,m}\right]$ are found, a backup procedure is detailed in Sect. 3.2 which expands the size of $\left[q_{i,1}, q_{i,m}\right]$ in order to prevent subsequent occurrences of extreme values. It must be pointed out that the probability of activating such backup procedure decreases as learning progresses, since the interval $[q_1, q_m]$ can only grow.

The simplicity of zeroth- and first-order piecewise polynomial activation functions (Eqs. 12 and 13) suggest that our model might be implemented with smaller computational cost on FPGAs, microcontrollers and other hardware with limited resources than standard activation functions because no transcendent functions are evaluated at any time, which ensures a low requirement of hardware resources [24].

The reported experimental results indicate that overfitting can be adequately controlled in our proposal. The average of the generalization ability obtained for the real classification problems with the proposed algorithms is very close to the value obtained with backpropagation for the least advantageous size of neural architecture for our proposals (five neurons in the hidden layer). Moreover, the performance of our models is significantly better in all data sets for the most advantageous size (two neurons). Finally, for the highest analyzed architecture size (ten neurons) the generalization ability of our proposals is better for the majority of the data sets (see Tables 6, 7).

For the Boolean problems the most advantageous architecture size for each algorithm depends dramatically on the data set. Hence, the generalization ability exhibits a large variability, being more stable and higher in terms of average performance for the proposed algorithms (see Tables 2, 3).

The average of the regression error calculated in terms of MSE is lower (better) for our proposals as it can be seen in Table 4. Also, for the majority of data sets and architecture configurations the regression error is lower for the proposed algorithms (Linear, Constant and Cubic), as seen in Table 8.

The performance of the networks for the proposed algorithms (Constant, Linear and Cubic) in terms of the accuracy at the time that the termination criterion is fulfilled is more stable (lower standard deviation) than that of the backpropagation algorithm with the different activation functions, as observed in Figs. 2 and 3. Moreover, the number of neurons of the neural architecture is also more stable for our proposals, as it can be observed in Fig. 4a. This indicates that our algorithms are more robust against overfitting.

The favorable results obtained by our nonparametric activation function approach, as compared to parametric activation functions, may be caused by a better usage of the neurons of the network. It must be taken into account that, by approximating the distribution function of the argument of the activation function by the activation function, the distribution of the output values of the neurons gets closer to a uniform distribution on the output range of the neurons. In this way, the amount of information carried by the output of each neuron is maximized. On the other hand, a parametric activation function might lead to similar output values, which would mean that the information content of the output is smaller.

The results of the generalization ability for the two data sets (Vertebral Column and Blood Transfusion) have also been analyzed by Castelli [7] who uses adaptive activation functions. Castelli's results show that they have got a significantly better generalization ability for these data sets although with larger standard deviations. This possibly due to a non-exhaustive search of the hyper-parameters. Moreover, our approach is more amenable to implementation

Table 6 Generalization ability in terms of accuracy (Acc) obtained with the proposed algorithms (Linear, Constant and Cubic) and with four different activation functions (Sig, Amp, ReLU and Shape), on a set of real-world problems from the UCI data set

Function	#A	#I	#N	Constant	Linear	Cubic	Sig	Amp	ReLU	Shape
Blood transfusion	5	748	5	79.70	80.84	79.54	77.84	78.11	76.62	76.43
			10	**81.84**	78.86	78.12	78.92	78.24	76.49	77.03
Cancer	9	286	5	97.32	97.34	97.76	95.85	95.69	96.03	96.52
			10	97.44	97.53	97.33	95.56	95.75	95.79	96.93
Card	51	690	5	85.90	86.12	86.69	84.39	83.71	85.15	82.54
			10	86.96	**86.90**	86.14	84.10	84.25	80.29	78.46
Climate	18	540	5	92.59	92.65	93.01	91.70	90.46	90.13	92.48
			10	93.00	93.00	**93.11**	91.65	90.46	90.65	92.37
Diabetes	8	768	5	76.59	76.96	76.54	75.78	75.34	70.99	69.41
			10	**77.09**	77.01	76.24	75.25	75.09	71.66	68.16
Fertility	10	100	5	87.10	87.00	**88.52**	85.90	87.00	87.00	88.00
			10	86.20	85.30	88.32	85.90	86.90	87.50	87.80
Heart	35	303	5	80.83	80.60	81.62	79.57	79.33	81.13	78.20
			10	81.20	**81.67**	80.98	79.70	77.80	78.70	74.33
Ionosphere	34	351	5	87.46	87.57	85.38	87.51	84.71	74.51	77.66
			10	88.83	87.57	**89.78**	89.23	89.66	78.4	75.97
Sonar	60	208	5	70.85	73.35	69.74	72.85	68.40	68.15	66.85
			10	75.50	73.95	72.26	**75.55**	72.90	66.40	66.45
Statlog heart	35	270	5	82.00	80.85	81.71	80.78	79.93	81.26	80.37
			10	81.70	**81.96**	**81.96**	80.04	80.11	80.00	77.44
Vertebral column	6	310	5	79.70	81.03	**86.76**	85.77	75.78	78.06	82.90
			10	81.84	84.55	81.33	85.16	77.74	80.01	72.58

Best results are shown in bold. See the main text for more details

on microcontrollers than Castelli's model due to the computational simplicity of our approach. This compensates for the relative lack of performance of our proposal.

In order to quantitatively assess the advantage of our approach, as compared to other learnable activation function approaches, we may compare the number of nodes m of our piecewise polynomial functions with the number of intervals required to approximate a sigmoid function (such as [7] employs) by means of piecewise Taylor approximation. A state-of-the-art Taylor approximation of this kind is proposed in [4], where up to 102 intervals are required to yield a good accuracy. In contrast to this, in Sect. 4 it is reported that our approach only needs $m = 20$ nodes in order to attain adequate performance. Hence, our approach requires about 5 times less memory space, since the memory requirements are proportional to the number of nodes/intervals. Moreover, our approach is also faster, since the table lookup for the relevant node/interval requires a computation time which is proportional to the base 2 logarithm of the number of nodes/intervals, if binary search is employed for maximum table lookup speed.

The number of epochs to fulfill the termination criterion is notably lower for the proposed algorithms than the Sig and Amp activation functions for any network size, quite similar to the ReLU activation function and significantly higher than the Shape activation function and for the majority of the functions (see Table 6).

Table 7 Generalization ability in terms of accuracy (Acc) obtained with the proposed algorithms (Linear, Constant and Cubic) and with four different activation functions (Sig, Amp, ReLU and Shape), on a set of Boolean functions from the MCNC benchmark

Function	#A	#I	#N	Constant	Linear	Cubic	Sig	Amp	ReLU	Shape
alu2k	10	1024	5	58.05	58.25	62.04	64.09	57.56	60.37	56.73
			10	58.50	58.47	61.90	**81.45**	61.75	66.50	57.34
alu2l	10	1024	5	59.12	57.21	61.28	64.52	56.89	62.08	56.29
			10	59.24	58.94	61.24	**70.96**	61.25	66.22	56.58
alu2m	10	1024	5	98.97	98.91	98.98	98.54	96.92	90.33	94.13
			10	**99.68**	99.65	99.54	98.96	98.86	96.46	89.94
alu2n	10	1024	5	**100.0**	**100.0**	**100.0**	99.00	98.99	99.64	95.92
			10	**100.0**	**100.0**	**100.0**	99.00	99.00	99.99	94.93
alu2o	10	1024	5	82.85	83.07	86.02	84.14	81.99	83.31	78.25
			10	82.81	83.59	**86.62**	86.54	85.12	84.07	76.53
alu2p	10	1024	5	94.05	91.96	92.15	97.95	95.71	95.93	95.01
			10	98.13	98.13	98.26	98.52	**98.55**	97.44	86.23
z4ml24T	7	128	5	93.50	89.83	93.24	80.08	77.58	83.25	76.50
			10	93.83	92.58	**94.03**	80.25	77.92	82.00	77.25
z4ml25T	7	128	5	53.75	53.83	55.69	43.42	46.75	51.83	45.92
			10	64.08	**65.17**	64.99	52.42	46.33	51.42	50.00
z4ml26T	7	128	5	62.75	59.17	60.38	45.92	49.50	57.83	48.17
			10	**75.08**	72.58	74.89	59.08	48.75	64.83	49.25
z4ml27T	7	128	5	63.25	65.00	62.26	55.25	53.08	62.08	52.97
			10	87.73	81.33	**88.10**	71.92	52.25	77.08	60.67

Best results are shown in bold. See the main text for more details

Finally, some limitations of our approach must be pointed out. Our proposal is restricted to one-output neuron networks. Functions with more than one output can be implemented by using the appropriate number of networks of the proposed kind. This implies that the resulting system would have a higher computational complexity, compared to a single network with several outputs. Moreover, important correlations among the outputs cannot be exploited by weight sharing in a multi-output network. Besides that, weight sharing might help to alleviate overfitting whenever the data are scarce. For these reasons, one of our future research lines is the development of an extension to our proposal with multiple outputs. At the current state of our research, another limitation is that only one hidden layer is considered. Future work includes the extension to multiple hidden layers.

6 Conclusions

A new feedforward neural network model has been proposed. The considered activation functions are piecewise polynomial. The model features a mechanism to learn the activation function of each hidden neuron independently, whose error reduction properties have been formally proved. This way, our proposal exhibits a greater flexibility to adapt the network to the training data. Experiments have been carried out for regression and classification applications. The results demonstrate that our approach is a valid alternative for supervised

Table 8 Regression ability in terms of mean squared error (MSE) obtained with the proposed algorithms (Linear, Constant and Cubic) and with four different activation functions (Sig, Amp, ReLU and Shape), on a set of real-world problems from the UCI data set

Function	#A	#I	#N	Constant	Linear	Cubic	Sig	Amp	ReLU	Shape
Airfoil	5	1503	5	0.0150	0.0150	0.0136	0.0138	0.0154	0.0549	0.0287
			10	0.0137	0.0136	**0.0131**	0.0135	0.0144	0.0471	0.0294
C_Power	4	956	5	0.0030	0.0029	0.0030	0.0042	0.0054	0.0137	0.0048
Plant			10	0.0028	**0.0027**	0.0028	0.0042	0.0059	0.0123	0.0070
Concrete	8	1030	5	0.0112	0.0104	0.0136	0.0100	0.0120	0.0115	0.0108
Com_Str			10	**0.0095**	0.0098	0.0116	0.0122	0.0105	0.0097	0.0109
Concrete	10	103	5	0.0180	0.0158	0.0106	0.0106	0.0284	**0.0068**	0.0103
Slump			10	0.0145	0.0132	0.0095	0.0110	0.0128	0.0097	0.0097
Energy	8	768	5	0.0057	**0.0054**	0.0058	0.0066	0.0105	0.0068	0.0063
EfficientA			10	**0.0054**	0.0056	0.0058	0.0064	0.0107	0.0058	0.0072
Energy	8	768	5	0.0080	0.0082	0.0081	0.0083	0.0101	0.0081	0.0080
EfficientB			10	**0.0073**	0.0079	0.0081	0.0083	0.0098	0.0077	0.0089
Forest	12	517	5	**0.0030**	**0.0030**	0.0032	0.0045	0.0044	0.0036	0.0034
Fire			10	0.0033	0.0037	0.0036	0.0045	0.0044	0.0042	0.0034
Housing	13	506	5	0.0116	0.0111	0.0107	0.0110	0.0163	0.0116	0.0139
			10	0.0116	0.0116	**0.0110**	0.0107	0.0147	0.0109	0.0161
Parkinson	21	5875	5	0.0033	0.0032	0.0048	0.0046	0.0050	0.0044	0.0037
Telemonit			10	0.0032	**0.0031**	0.0041	0.0046	0.0053	0.0040	0.0038
Wine quality	11	1599	5	0.0184	0.0178	0.0179	0.0182	0.0193	0.0213	0.0187
Red			10	0.0177	0.0176	**0.0172**	0.0182	0.0195	0.0211	0.0189
Wine quality	11	4898	5	0.0167	0.0165	0.0171	0.0168	0.0172	0.0180	0.0170
White			10	0.0160	**0.0158**	0.0169	0.0167	0.0171	0.0176	0.0172
Yacht	6	308	5	0.0033	**0.0017**	0.0016	0.0079	0.0131	0.0081	0.0207
Hydrody			10	0.0059	0.0038	0.0026	0.0065	0.0122	0.0072	0.0186

Best results are shown in bold. See the main text for more details

learning tasks. This paper is a preliminary work; future works include the extension of this approach to neural networks with more than one output neuron and more than one hidden layer. This might be accomplished by employing functions which are differentiable with respect to the node locations and fast to evaluate, so that the chain rule can be applied to multiple neural layers and multiple outputs.

Acknowledgements This work is partially supported by the Ministry of Economy and Competitiveness of Spain under Grants TIN2014-53465-R, project name Video surveillance by active search of anomalous events, and TIN2014-57341-R, project name Metaheuristics, holistic intelligence and smart mobility. It is also partially supported by the Autonomous Government of Andalusia (Spain) under project P12-TIC-657, project name Self-organizing systems and robust estimators for video surveillance. All of them include funds from the European Regional Development Fund (ERDF). The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the SCBI (Supercomputing and Bioinformatics) center of the University of Málaga. They also gratefully acknowledge the support of NVIDIA Corporation with the donation of two Titan X GPUs used for this research.

Appendix

Proof of Proposition 2 Let us assume that $u_{r,i} \in \left[q_{i,k}, q_{i,k+1}\right)$. Therefore,

$$g_i\left(u_{r,i}\right) = \frac{k + \delta_{i,k}\left(u_{r,i}\right)}{m} \tag{63}$$

where $\delta_{i,k}\left(u_{r,i}\right) \in [0, 1]$. The exact form of $\delta\left(u_{r,i}\right)$ depends on the order of the polynomials γ.

If $w_{2,i}\left(y_r - z_r\right) < 0$ and $\left|\frac{2w_{2,i}}{m}\right| < \lambda \left|y_r - z_r\right|$, then the update Eq. (36) implies that:

$$q_{i,k} \leq \bar{q}_{i,k+1} \leq u_{r,i} \tag{64}$$

Therefore,

$$\bar{g}_i\left(u_{r,i}\right) = \frac{k + 1 + \bar{\delta}_{i,k+1}\left(u_{r,i}\right)}{m} \tag{65}$$

where the bars correspond to the values obtained after executing the update. Moreover, from (4):

$$\begin{aligned} y_r - \bar{y}_r &= \sum_{s=1}^{L} w_{2,s} g_s\left(u_{r,s}\right) - \sum_{s=1}^{L} w_{2,s} \bar{g}_s\left(u_{r,s}\right) \\ &= w_{2,i} g_i\left(u_{r,i}\right) - w_{2,i} \bar{g}_i\left(u_{r,i}\right) \end{aligned} \tag{66}$$

Then from (63), (65) and (66):

$$\begin{aligned} y_r - \bar{y}_r &= w_{2,i}\left(\frac{k + \delta_{i,k}\left(u_{r,i}\right)}{m} - \frac{k + 1 + \bar{\delta}_{i,k+1}\left(u_{r,i}\right)}{m}\right) \\ &= w_{2,i} \frac{\delta_{i,k}\left(u_{r,i}\right) - \bar{\delta}_{i,k+1}\left(u_{r,i}\right) - 1}{m} \end{aligned} \tag{67}$$

Since $w_{2,i}\left(y_r - z_r\right) < 0$, there are two possible cases: (a) $\left(w_{2,i} > 0\right) \wedge \left(y_r - z_r < 0\right)$; (b) $\left(w_{2,i} < 0\right) \wedge \left(y_r - z_r > 0\right)$.

For case (a), since $\delta_{i,k}\left(u_{r,i}\right), \bar{\delta}_{i,k+1}\left(u_{r,i}\right) \in [0, 1]$, from (67) we obtain:

$$-\frac{2w_{2,i}}{m} \leq y_r - \bar{y}_r \leq 0 \tag{68}$$

On the other hand, since $y_r - z_r < 0$, $w_{2,i} > 0$, $\lambda \in (0, 1]$ and $\left|\frac{2w_{2,i}}{m}\right| < \lambda \left|y_r - z_r\right|$ we have:

$$y_r - z_r \leq -\frac{2w_{2,i}}{m} \tag{69}$$

From (68) and (69):

$$y_r - z_r \leq y_r - \bar{y}_r \leq 0 \tag{70}$$

$$-z_r \leq -\bar{y}_r \leq -y_r \tag{71}$$

$$y_r \leq \bar{y}_r \leq z_r \tag{72}$$

$$\bar{E}_r \leq E_r \tag{73}$$

That is, the new squared error $\bar{E}_r$ is lower than or equal to the old squared error E_r, as required.

For case (b), since $\delta_{i,k}\left(u_{r,i}\right), \bar{\delta}_{i,k-1}\left(u_{r,i}\right) \in [0, 1]$, from (67) we obtain:

$$0 \leq y_r - \bar{y}_r \leq -\frac{2w_{2,i}}{m} \tag{74}$$

On the other hand, since $y_r - z_r > 0$, $w_i^2 < 0$, $\lambda \in (0, 1]$ and $\left|\frac{2w_{2,i}}{m}\right| < \lambda \left|y_r - z_r\right|$ we have:

$$-\frac{2w_{2,i}}{m} \leq y_r - z_r \tag{75}$$

From (74) and (75):

$$0 \leq y_r - \bar{y}_r \leq y_r - z_r \tag{76}$$

$$-y_r \leq -\bar{y}_r \leq -z_r \tag{77}$$

$$z_r \leq \bar{y}_r \leq y_r \tag{78}$$

$$\bar{E}_r \leq E_r \tag{79}$$

And again the new squared error $\bar{E}_r$ is lower than or equal to the old squared error E_r, as required. □

References

1. Agostinelli F, Hoffman M, Sadowski PJ, Baldi P (2014) Learning activation functions to improve deep neural networks. CoRR arXiv:1412.6830, URL http://arxiv.org/abs/1412.6830
2. Barron AR (1993) Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans Inf Theor 39(3):930–945
3. Bartlett PL, Maiorov V, Meir R (1998) Almost linear VC-dimension bounds for piecewise polynomial networks. Neural Comput 10(8):2159–2173
4. Campo ID, Finker R, Echanobe J, Basterretxea K (2013) Controlled accuracy approximation of sigmoid function for efficient FPGA-based implementation of artificial neurons. Electron Lett 49(25):1598–1600
5. Castelli I, Trentin E (2012a) Semi-unsupervised weighted maximum-likelihood estimation of joint densities for the co-training of adaptive activation functions. In: Schwenker F, Trentin E (eds) Partially supervised learning: first IAPR TC3 workshop, PSL 2011, Ulm, 15–16 Sept 2011. Revised selected papers, Springer, Berlin, Heidelberg, pp 62–71
6. Castelli I, Trentin E (2012b) Supervised and unsupervised co-training of adaptive activation functions in neural nets. In: Schwenker F, Trentin E (eds) Partially supervised learning: first IAPR TC3 workshop, PSL 2011, Ulm, 15–16 Sept 2011. Revised selected papers, Springer, Berlin, Heidelberg, pp 52–61
7. Castelli I, Trentin E (2014) Combination of supervised and unsupervised learning for training the activation functions of neural networks. Pattern Recognit Lett 37(Supplement C):178–191
8. Chen CT, Chang WD (1996) A feedforward neural network with function shape autotuning. Neural Netw 9(4):627–641
9. Costarelli D, Vinti G (2016) Max-product neural network and quasi-interpolation operators activated by sigmoidal functions. J Approx Theory 209:1–22
10. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2(4):303–314
11. Ertugrul ÖF (2018) A novel type of activation function in artificial neural networks: trained activation function. Neural Netw 99:148–157
12. Fritsch FN, Carlson RE (1980) Monotone piecewise cubic interpolation. SIAM J Numer Anal 17:238–246
13. Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics (AISTATS 2011)
14. Goodfellow IJ, Warde-Farley D, Mirza M, Courville AC, Bengio Y (2013) Maxout networks. In: Proceedings of the 30th international conference on machine learning, ICML 2013, Atlanta, 16–21 June 2013, pp 1319–1327
15. Gulcehre C, Cho K, Pascanu R, Bengio Y (2014) Learned-norm pooling for deep neural networks. Lect Notes Comput Sci 8724:530–546
16. Hawkins DM (2004) The problem of overfitting. J Chem Inf Comput Sci 44(1):1–12

17. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
18. Huynh HT, Won Y (2009) Extreme learning machine with fuzzy activation function. In: 2009 Fifth international joint conference on INC, IMS and IDC. https://doi.org/10.1109/NCM.2009.206
19. Kang M, Palmer-Brown D (2005) An adaptive function neural network (ADFUNN) for phrase recognition. In: IEEE international joint conference on neural networks, 2005. IJCNN 2005, vol 1, pp 593–597
20. Kang M, Palmer-Brown D (2007) A multi-layer adaptive function neural network (MADFUNN) for letter image recognition. In: International joint conference on neural networks, 2007. IJCNN 2007, pp 2817–2822
21. Maas AL, Hannun AY, Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. In: 30 th International conference on machine learning, vol 28
22. Microelectronics Center of North Carolina (2016) MCNC benchmarks. http://www.cbl.ncsu.edu:16080/benchmarks/. Accessed 15 Oct 2016
23. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10), pp 807–814
24. Ortega-Zamorano F, Jerez J, Juarez G, Perez J, Franco L (2014) High precision fpga implementation of neural network activation functions. In: IEEE symposium on intelligent embedded systems (IES), 2014, pp 55–60. https://doi.org/10.1109/INTELES.2014.7008986
25. Rumelhart D, Hinton G, Williams R (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
26. Rumelhart DE, Hinton GE, Williams RJ (1988) Learning representations by back-propagating errors. In: Anderson JA, Rosenfeld E (eds) Neurocomputing: foundations of research. MIT Press, Cambridge, pp 696–699
27. Sakurai A (1998) Tight bounds for the VC-dimension of piecewise polynomial networks. In: Advances in neural information processing systems, vol 11, pp 323–329
28. Springenberg J, Riedmiller M (2013) Improving deep neural networks with probabilistic maxout units, pp 1–10. arXiv:1312.6116
29. Sunat K, Lursinsap C, Chu CHH (2007) The p-recursive piecewise polynomial sigmoid generators and first-order algorithms for multilayer tanh-like neurons. Neural Comput Appl 16(1):33–47
30. Trentin E (2001) Networks with trainable amplitude of activation functions. Neural Netw 14(4–5):471–493
31. University of California Irvine (2016) Machine learning repository. http://archive.ics.uci.edu/ml/. Accessed 17 Oct 2016
32. Vecci L, Piazza F, Uncini A (1998) Learning and approximation capabilities of adaptive spline activation function neural networks. Neural Netw 11(2):259–270
33. Wang GT, Li P, Cao JT (2012) Variable activation function extreme learning machine based on residual prediction compensation. Soft Comput 16(9):1477–1484. https://doi.org/10.1007/s00500-012-0817-5
34. Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University
35. Zhang M, Fulcher J, Scofield RA (1997) Rainfall estimation using artificial neural network group. Neurocomputing 16(2):97–115