Fast Feedforward Neural Networks
with Diffused Nonlinear Weight Functions

Artur Rataj, e–mail arataj@iitis.gliwice.pl
Institute of Theoretical and Applied Computer Science
of the Polish Academy of Sciences, Bałtycka 5, Gliwice, Poland

Abstract

In this paper, feedforward neural networks with nonlinear weight functions based on look–up tables tables are presented. Special low–cost regularization, which heuristically 'diffuses' values from the more important regions of the look–up tables to the less important regions of the look–up tables, is used during training of the neural networks. This way the presented networks may in some applications have both high flexibility and high generalization quality, while retaining a relatively low number of connections.

keywords: feedforward neural networks, nonlinear weight functions, generalization, diffusion

1 INTRODUCTION

It is long know that a substantial amount of adaptive nonlinear processing of signals in biological neurons occurs in synaptic connections (Grossberg, 1969; Thomson and Deuchars, 1994; Abbott et al., 1997). This, roughly speaking, gives a densely connected biological neural network the power of the order of N^2 of adaptive nonlinear processing units, where N is the number of nodes. For large N, it can be a substantial difference in comparison to N adaptive nonlinear processing units in the classic feedforward artificial neural networks, where a single connection can only process a signal in a linear way. Keeping the adaptable weights linear may lead to the need of networks with an enormous number of nodes and connections, what in effect may make these networks extremely slow. The other problem with artificial neural networks with linear weight functions may be the relatively strong 'artifacts' that can be introduced into the generalizing function if the training set has very nonlinear patterns. A classic example of such a set is the 'two spirals' benchmark (Lang and Witbrock, 1988; Fahlman and Lebiere, 1990).

In this paper, feedforward neural networks with nonlinear weight functions based on look–up tables tables are proposed. Such networks can have a relatively high ratio of flexibility to the number of connections. Special regularization algorithm that unevenly 'diffuses' the weight functions is introduced to reduce negative effects on generalization quality, that may be caused by the high flexibility of the presented neural networks. The presented neural networks use continuous output value neurons, yet the presented algorithm may be easily extended to other types of neurons, i. e. the spiking neurons.

There were various feedforward neural networks proposed with nonlinear adaptive functions. Piazza et al. (1993), Uncini et al. (1998) and Trentin (2001) presented neural networks with nonlinear adaptive activation functions. Either in the case of using adaptive nonlinear weight functions, like in the case of the networks introduced in this paper, or in the case of using adaptive nonlinear activation functions, the flexibility of a neural network increases. However, in the latter case there is an additional constraint, that

there is a single nonlinear adaptive function for each neuron, and not for each connection. Another approach, where a product of signals has a separate weight, was proposed by Koch and Poggio (1992) and Gurney (1992a,b). For example, Gurney introduced neurons with the following function

$$y = \sum_{k=1}^{N} w_k \prod_{i \in I_k} x_i \qquad (1)$$

where x_i are inputs of the neuron, y is the neuron output, w_k are the weights of the neuron and I_k is the kth in a series of sets such that each of them contains one of the possible 2^n selections of the first n integers. Such neurons could provide a relatively complex pattern that would their inputs to their output values, yet a dependency between a single input x_i and the output y, assuming the rest of inputs is constant, is still linear.

The presented neural networks have the quality of choosing a subset of adaptive parameters during a single propagation process. If a signal is propagated through a look–up table based weight function, only one or two of the parameters of the function are used to compute its value. This enables the presented networks to have a large number of adaptive parameters, and still be relatively fast. This is in contrast to the cases of the networks where each adaptive parameter of an adaptive function changes the adaptive function in its whole domain, as it is in the mentioned networks presented by Koch and Poggio (1992) and Gurney (1992a,b), or as it is in the networks with adaptive activation functions proposed by Chen and Chang (1996).

The higher flexibility of the networks with adaptive look–up table based functions, either the weight or activation ones, can worsen generalization. The direct use of coefficients from a look–up table, further denoted by LUT, proposed by Piazza et al. (1993), caused serious problems because of the lack of smoothness of the activation functions. It can be avoided by using smooth functions like the polynomial functions, and therefore by limiting the flexibility of a neural network. Such networks have been presented by Wang et al. (1994), Uncini et al. (1998), Lorenzo et al. (1998) and Guarnieri and Piazza (1999). Such a limitation of flexibility, however, can cause the same problems as in the case of the classic feedforward neural networks – again, a large number of connections may be needed. In contrast, in the presented method the weight functions can possibly be very flexible, but a regularization, called diffusion, is used, which is dependable not only on the values of the weight functions, but adapts to the density of occurrence of different values of signals propagated through the network. This way, in the regularization values of the weight functions are heuristically 'diffused' from the more important regions of the LUTs to the less important regions of the LUTs.
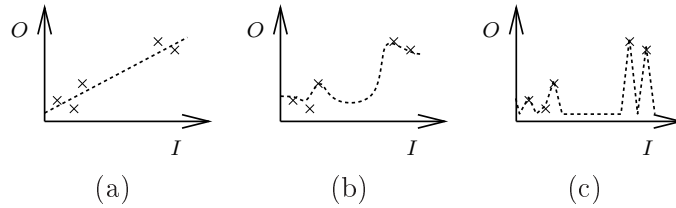

Figure 1: A schematic example of generalization using (a) linear, (b) smooth and (c) LUT weight function types. Crosses schematically denote observation attributes propagated from inputs of the network and respective values that minimize propagated derivatives of the training error function.

2

Each nonlinear weight function in the presented neural networks has an accompanying visit table determining the visit function. The visit function represents the weighted mean values connected with the densities of occurrences of arguments of the weight function during propagation of attributes of observations in the learning process. The more recently in a training process a given value has been propagated, the greater, approximately, is its importance in the mean computation. During the training process, there is a 'diffusion' of values of the look–up table weight function, from regions with the higher respective values of the visit function to regions with the lower respective values of the visit function. The visit function is also diffused, like the weight function. Therefore, values of the weight function, diffusing from a given 'source', 'carry' with them information about the source 'intensity'.

There are schematic examples of generalization for the cases of the linear, spline–like and LUT weight functions in Fig. 1. Problems with generalization resulting from high flexibilities of the two latter weight functions are clearly seen. There is also a schematic example of diffusion of a LUT weight function O and its visit function V in some subsequent iterations in Fig. 2, with diffusion. In both of these figures, I is the argument of a weight function. It can be seen from the example, comparing it to
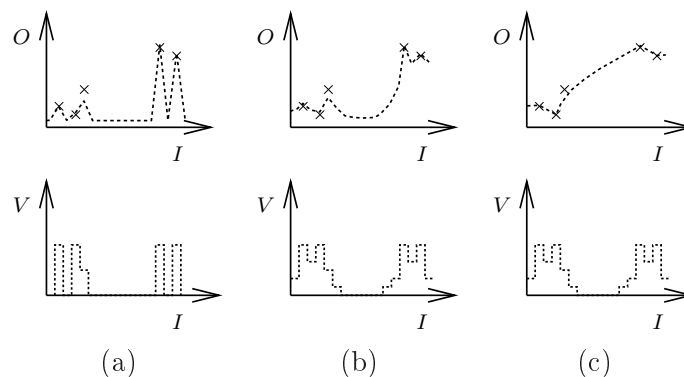

Figure 2: A schematic example of generalization using diffused LUT weight function – (a) an iteration i, (b) an iteration j, $j > i$, (c) an iteration k, $k > j$. Crosses schematically denote observation attributes propagated from inputs of the network and respective values that minimize propagated derivatives of the training error function. There is an according visit function shown for each diagram of the weight function.

Fig. 1(c), that the diffusion can improve the quality of generalization.

2 STRUCTURE OF THE NETWORKS

Let us separate the notion of a neuron into a number of connections and a node. We do so because there are two different types of connections in the discussed neural networks. The weight function is associated with a given connection, and the combination and activation functions are associated with a node. Let n denote the iteration number.

In a connection i, the computation of the connection output value $O_i(n)$ on the basis of the connection input value $I_i(n)$ is done by using the connection weight function.

The combination function $u_k^c(n)$ of a node k sums its arguments, like in the classic

McCulloch and Pitts (1943) type neurons

$$u_k^c(n) = \sum_{i \in M} O_i(n), \tag{2}$$

where M is a set of indexes of the input connections of the node k.

The activation function u_k^a of a node k is sigmoid–like

$$y_k(n) = u_k^a(u_k^c(n)) = \tanh\left(u_k^c(n)\right), \tag{3}$$

where $y_k(n)$ is the output value of the node k. The activation function softly clamps the combination function value, so that the value fits into the domains of the LUT weight functions.

A linear connection i has a weight function of the form

$$O_i(n) = w_s^i I_i(n), \tag{4}$$

where w_s^i is the connection scalar weight.

A LUT connection i has the following weight function

$$O_i(n) = w_l^i(n) I_i(n) + r\left(\mathbf{w}_r^i(n), I_i(n)\right), \tag{5}$$

where $w_l^i(n) I(n)$ is a component further called the linear one and $r(\mathbf{w}_r^i(n), I_i(n))$ is a component further called the LUT one. The coefficients $w_l^i(n)$ and $\mathbf{w}_r^i(n)$ are the parameters of the connection weight function. The parameter $w_l^i(n)$ is a scalar and the parameter $\mathbf{w}_r^i(n)$ is the LUT of the weight function. The function $r(\mathbf{w}_r^i(n), I(n))$ is determined by a curve being a linear interpolation of several points $p_j^i(n)(I^j, w_{r,j}^i(n))$, where $j = 0 \ldots r_{\mathrm{res}} - 1$ and $w_{r,j}^i(n)$ is a jth element of $\mathbf{w}_r^i(n)$. The first coordinate of points on the curve denotes arguments and the second one values of the function. The values I^j are equally distributed values as follows

$$I^j = I_{\min} + \frac{j}{r_{\mathrm{res}} - 1}(I_{\max} - I_{\min}). \tag{6}$$

We will call r_{res} the LUT component resolution. The coefficients $I_{\min}$ and $I_{\max}$ are $I_i(n)$ minimum and maximum allowable values, respectively. These values are equal to the minimum and maximum values of the activation functions. Let the function $r(\mathbf{w}_r^i(n), I(n))$ is computed using the following piece–wise linear interpolation:

$$r\left(\mathbf{w}_r^i(n), I(n)\right) = \left(\lfloor S(n) \rfloor + 1 - S(n)\right) w_{r,\lfloor S(n) \rfloor}^i(n) + \left(S(n) - \lfloor S(n) \rfloor\right) w_{r,\lfloor S(n) \rfloor + 1}^i(n)$$

$$S(n) = \frac{I(n) - I_{\min}}{I_{\max} - I_{\min}} \left(r_{\mathrm{res}} - 1\right). \tag{7}$$

Alternatively, of course, another interpolation type, for example cubic spline interpolation (Greville, 1969), could be used.

The linear component $w_l^i(n) I_i(n)$ has the role of generalizing linear patterns. It has been found in tests, that a neural network with both the linear and the LUT components may in some cases perform substantially better than a network with only one type of the components.

The introduced networks are fully connected multilayer feedforward neural networks (Bishop, 1995; Hertz et al., 1991).

All linear connections that would be used in a basic layered feedforward neural network, except of these from bias elements, are replaced with LUT connections in the presented networks. Bias elements have a constant output, and therefore there is no need for a LUT connection.

4

3   TRAINING

On–line training with error backpropagation (Bishop, 1995; Hertz et al., 1991) is used.

Let a training set be given. Each observation k in the set has i argument attributes and j value attributes $\left\{ x_0^k, x_1^k, \ldots x_{i-1}^k, d_0^k, d_1^k, \ldots d_{j-1}^k \right\}$ and we want the network to generalize the relation between the attributes with a function mapping the argument attributes to the value attributes. Let e_i be an error function derivative backpropagated to the connection i, and μ be the learning step.

The training algorithm is somewhat mixed the regularization algorithm. To keep their descriptions relatively separate, we will only shortly mention now the regularization functions $R_d(w)$ and $R_s(w, \Delta w)$ and when the regularization is performed. The function $R_d(w)$ is a simple weight decay (Krogh and Hertz, 1992) – its value is its argument multiplied by a value in the range $(0, 1\rangle$. The function $R_s(w, \Delta w)$ is a kind of an indirect weight decay, that does not have some drawbacks of the regular weight decay, but let us now for simplicity assume that

$$R_s(w, \Delta w) = \Delta w. \tag{8}$$

Booth of these functions will be described in detail in Sec. 4. The regularization of the LUT component can be much more computationally complex that the regularization of the linear connection and of the linear component, because of the number of adaptive parameters of the LUT component. Because of this, while the regularization of the linear connection and of the linear component is performed in every training iteration, the regularization of the LUT component is performed only in some training iterations, in which the following is true:

$$\zeta > \zeta_i(n), \tag{9}$$

where

$$\zeta_i(n) = \mathrm{rand}(0.0, 1.0), \tag{10}$$

and the function $\mathrm{rand}(0, 1)$ is a uniform random number generator which returns a random value within $\langle 0, 1)$. The more the ζ coefficient is lower than 1, the lesser is mean computation complexity per training iteration, but the quality of the regularization may worse. The LUT regularization can accordingly be 'stronger' to counterbalance its exclusion in some iterations. The random exclusion (9), instead of a regular one, is used to rule out the possible resonance with the training observations. The condition (9) will repeat in some equations in the further sections.

3.1   UPDATING WEIGHTS OF THE LINEAR CONNECTIONS

Let there be a linear connection i with the input value $I_i(n)$. The weight of the linear connection is adapted as in the classic propagation, with the weight decay applied:

$$w_s^i(n+1) = R_d\Big(w_s^i(n) + \Delta w_s^i(n) \Big), \tag{11}$$

where

$$\Delta w_s^i(n) = -R_s\Big(w_s^i(n), \mu e_i(n) I_i(n) \Big). \tag{12}$$

3.2   UPDATING WEIGHTS OF THE LUT CONNECTIONS

A LUT component is initialized before the training of the neural network as a random linear function with relatively low values and a relatively low derivative.

Let there be a LUT connection i. Let the linear component weight $w_l^i(n)$ be adapted analogously to the weight in the linear connection:

$$w_l^i(n+1) = R_d\Big(w_l^i(n) + \nu\Delta w_l^i(n)\Big), \tag{13}$$

where

$$\Delta w_l^i(n) = -R_s\Big(w_l^i(n), \mu e_i(n) I_i(n)\Big). \tag{14}$$

The coefficient ν specifies a relation of the adaptation speed of the linear component to the adaptation speed of the LUT component. Increasing the value may cause the linear patterns to have a greater impact on the generalizing function.

Let the LUT component weight $\mathbf{w}_r^i(n)$ be adapted also in a similar way of that of the linear connection:

$$r\Big(\mathbf{w}_r^{i**}(n+1), I_i(n)\Big) = r\Big(\mathbf{w}_r^i(n), I_i(n)\Big) + \Delta w_r^i(n)$$
$$r\Big(\mathbf{w}_r^{i*}(n+1), I_i(n)\Big) = \begin{cases} R_d\Big(r\big(\mathbf{w}_r^{i**}(n), I_i(n)\big)\Big) & \text{if } \zeta > \zeta_i \\ r\Big(\mathbf{w}_r^{i**}(n), I_i(n)\Big) & \text{if } \zeta \le \zeta_i \end{cases}, \tag{15}$$

where

$$\Delta w_r^i(n) = -R_s\Big(r\big(\mathbf{w}_r^i(n), I_i(n)\big), \mu e_i(n)\Big). \tag{16}$$

Because the value of the LUT component is not a product of I_i and of $r(\mathbf{w}_r^i(n), I_i(n))$, but it is the value $r(\mathbf{w}_r^i(n), I_i(n))$ itself, there is the term $e_i(n)$ in (16) instead of the term $e_i(n) I_i(n)$ as in (12). The symbol * in (16) and further in this section denotes a value before the diffusion of the LUT component, that will be described in Sec. 4.2.

The equation (15) changes a value of the LUT component, but it does not decompose the change on the individual values in the LUT. Let the following conditions be given on the update of the LUT. If $I_i(n)$ is equal to a I^j coordinate of an approximated point $p_j^i(I^j, w_{r,j}^i(n))$, then only that point value $w_{r,j}^i(n)$ is changed. To fulfill (15),

$$w_{r,j}^{i**}(n+1) = w_{r,j}^i(n) + \Delta w_r^i(n). \tag{17}$$

Otherwise, $\exists j \in \langle 0, r_{\text{res}} - 2 \rangle$, $I_i(n) \in (I^j, I^{j+1})$. Then, values $w_{r,j}^i$ and $w_{r,j+1}^i$ are changed, using such amounts $\Delta w_L^r(n)$ and $\Delta w_H^r(n)$, respectively,

$$\begin{aligned} w_{r,j}^{i**}(n+1) &= w_{r,j}^i(n) + \Delta w_L^r(n) \\ w_{r,j+1}^{i**}(n+1) &= w_{r,j+1}^i(n) + \Delta w_H^r(n) \end{aligned}, \tag{18}$$

that

$$\frac{\Delta w_H^r(n)}{\Delta w_L^r(n)} = \frac{I_i(n) - I^j(n)}{I^{j+1}(n) - I_i(n)}. \tag{19}$$

Therefore, one of the modified points, let it be noted k, whose I^k value is possibly nearer to $I_i(n)$, has its $w_{r,k}^i$ value changed by a greater amount. To fulfill (15) and (19), $\Delta w_L^r(n)$ and $\Delta w_H^r(n)$ have the following equations

$$\begin{aligned} \Delta w_L^r(n) &= \Delta w_r^i(n) \frac{r_L}{2r_L^2 - 2r_L + 1} \quad r_L = I_{j+1}^i(n) - I_i(n) \\ \Delta w_H^r(n) &= \Delta w_r^i(n) \frac{r_H}{2r_H^2 - 2r_H + 1} \quad r_H = I_i(n) - I_j^i(n) \end{aligned}. \tag{20}$$

3.3 APPROXIMATED DERIVATIVE OF LUT WEIGHT FUNCTION

For error backpropagation to work, a derivative of a weight function in respect to a connection input value is needed. In the case of a LUT connection, an approximation of the derivative will be used instead. Let it have an equation as follows

$$\frac{\partial v\Big(\mathbf{w}_i(n), I_i(n)\Big)}{\partial I_i(n)} = c_l^i(n) + c_r^i(n), \tag{21}$$

where $c_l^i(n)$ and $c_r^i(n)$ are a derivative of the linear component and approximated derivative of the LUT component, respectively. Of course,

$$c_l^i(n) = w_l^i(n). \tag{22}$$

In the case of $c_r^i(n)$, a derivative approximation evaluated as the difference of neighboring LUT values is not used, because it could be too sensitive to individual weights changes and could cause numerical instability (Guarnieri and Piazza, 1999). Instead, the approximated derivative of a LUT component r of a connection i had an equation as follows

$$c_r^i(n) = \parallel A \parallel^{-1} \sum_{a \in A} \frac{r\Big(\mathbf{w}_r^i, c(I_i(n) + a)\Big) - r\Big(\mathbf{w}_r^i, c(I_i(n) - a)\Big)}{c(I_i(n) + a) - c(I_i(n) - a)}$$

$$A = \{a_l, a_m a_l, a_m^2 a_l, \dots a_f\} \quad a_f \leq a_h \quad a_m a_f > a_h \tag{23}$$

$$c(q) = \begin{cases} I_{\min} & \text{if } q < I_{\min} \\ q & \text{if } I_{\min} \leq q \leq I_{\max} \\ I_{\max} & \text{if } q > I_{\max} \end{cases}$$

Therefore, the approximated derivative is the mean of several differential ratios of the LUT component. The coefficient a_l is the minimum value of a, the coefficient a_h is the maximum value of a that possibly exists, and a_m determines the number of the ratios.

4 REGULARIZATION

Two types of regularization are used in the proposed neural networks – of absolute values of weight functions and and of diffusion of the LUT components.

4.1 REGULARIZATION OF ABSOLUTE VALUES OF WEIGHT FUNCTIONS

This type of regularization is a kind of weight decay, that tries to prevent absolute values of the weight functions from getting too large. Weight decay can improve generalization (Krogh and Hertz, 1992). The regularization is used for the adaptable parameter w_s^i of a linear connection, the linear component adaptable parameter w_l^i and the LUT component adaptable parameters $\mathbf{w}_r^i$. To regularize weights, the functions $R_s(w, \Delta w)$ and $R_d(w)$ are used. The functions were already used in the equations in Sec. 3. In this section, the functions will be described in more detail.

Let the function $R_d(w)$ be as follows

$$R_d(w) = (1 - R_b^s)w. \tag{24}$$

As it can be seen, it is a simple weight decay, whose strength is determined by the coefficient R_b^s.

A weight decay like in (24) may have the disadvantage of preventing the training process of converging exactly into a local minimum – the decay always 'pushes' the weights toward zero. Yet this type regularization can still be very important (Krogh and Hertz, 1992). In the presented algorithm, the following solution is proposed. The weight decay (24), should it be required to be too strong, is partially substituted by another type of weight decay, that operates not directly on the values of weights, but instead on the gains of the weights. Let the another type of weight decay be represented by the function $R_s(w, \Delta w)$, that has the following equation:

$$R_s(w, \Delta w) = \begin{cases} \dfrac{\exp(R_a^s w \Delta w) - 1}{R_a^s w} & \text{if } w \neq 0 \wedge R_a^s \neq 0 \\ \Delta w & \text{if } w = 0 \vee R_a^s = 0 \end{cases} \tag{25}$$

where R_a^s determines the level of the regularization. As it can be seen, the more w is positive, the more the ratio $\frac{R_s(w, \Delta w)}{\Delta w}$ decreases as Δw increases, and conversely, the more w is negative, the more the ratio $\frac{R_s(w, \Delta w)}{\Delta w}$ increases as Δw increases. That type of regularization may both slow down increasing of absolute values of the weight functions and speed up decreasing the values.

4.2   DIFFUSION OF THE LUT COMPONENT

The idea behind diffusion is that an observation has a 'significance' and while its attributes are propagated through the network, the 'significance' is marked in respective regions of the weight functions. In the diffusion process, values in the more 'significant' regions of the weight functions are heuristically 'diffused' to the less 'significant' regions of the weight function.

The algorithm of diffusion was constructed so to have low memory requirements and low time complexity. For each LUT component it needs only one additional table of the size of LUT of the component, and the time complexity is roughly linear to the resolution of the LUT. On the other hand, the algorithm is very far from the Fick's diffusion equation, and the 'significance' estimation is very heuristic and rather crude. The algorithm, however, keeps the time of a single iteration relatively short. This may be important if there are many training observations, and in effect many iterations to propagate the attributes of the observations are required.

In the process of diffusion, roughly speaking, weighted density of occurrence of LUT component arguments, at different regions of the LUT component domain, is computed. In computing of the density, there is a higher importance given to the more recent iterations. There are two reasons for giving the more recent iterations a higher importance. First, because of the adaptation of weight functions during training, the way of propagation of signals may gradually change, and we want the weight functions to fit to the more 'current' way of propagation of the signals. Secondly, because we use an on–line learning method, there may be possible trends in the training data.

Values related to the densities are stored in the visit table. For each single value in LUT of the weight function, there is a single corresponding value in the visit table. Values within the weight function LUT having the relatively higher corresponding values in the visit table are 'diffused' to these neighboring ones that have the relatively lower corresponding values in the visit table. Let there be the mean m of two such neighboring values of the weight function LUT before diffusion. After the diffusion, the value, that had

the higher corresponding value in a visit table, moves less towards m than the other value. The diffusion is also applied to the visit table. This is because if a value 'diffuses' to a neighboring one, an 'importance' of the value 'diffuses' also There is a schematic example of generalization using a diffused LUT weight function in Fig. 2. The diffusion 'spans' incrementally the LUT component function in regions relatively less frequently modified or not modified at all, where the 'spanning' regions are these relatively more frequently modified. There is no binary division only into extreme 'spanning' and 'spanned' regions, of course, as the visit functions are multivalued.

During the diffusion process, the LUT functions are also 'smoothed' by decreasing absolute differences between neighboring values in the LUTs, to reduce the problems caused by the lack of smoothness as reported by Piazza et al. (1993).

Let there be two subsequent values $w_{r,j}^{i*}(n+1)$ and $w_{r,j+1}^{i*}(n+1)$ of a LUT component $r(\mathbf{w}_r^{i*}(n+1), I_i(n))$, as described in Sec. 3. We want to smooth $r(\mathbf{w}_r^{i*}(n+1), I_i(n))$ by making absolute difference $|w_{r,j+1}^i(n+1) - w_{r,j}^i(n+1)|$ smaller than $|w_{r,j+1}^{i*}(n+1) - w_{r,j}^{i*}(n+1)|$. We also possibly want to 'diffuse' each $w_{r,j}^{i*}(n+1)$ value to $w_{r,j-1}^i(n+1)$ and $w_{r,j+1}^i(n+1)$, depending on visit table values. Let a LUT element $w_{r,j}^i(n)$, $j = 0 \ldots r_{\text{res}} - 1$, have its associated visit table element $V_j^i(n)$. Let $V_j^{i*}(n+1)$ be the visit function values before LUT component regularization. The following equation fulfills the discussed assumptions:

$$w_{r,j}^{i,L}(n+1) = m_j - \frac{\tanh(R_a^r d_j)}{2R_a^r(1 + R_b^r p_j)} \quad 0 \leq j \leq r_{\text{res}} - 2$$

$$w_{r,j+1}^{i,H}(n+1) = m_j + \frac{\tanh(R_a^r d_j)}{2R_a^r(1 + R_b^r p_j^{-1})} \quad 0 \leq j \leq r_{\text{res}} - 2$$

$$d_j = w_{r,j+1}^{i*}(n+1) - w_{r,j}^{i*}(n+1)$$

$$m_j = \left(w_{r,j}^{i*}(n+1) + w_{r,j+1}^{i*}(n+1)\right)/2$$

$$p_j = \frac{V_{j+1}^{i*}(n+1)}{V_j^{i*}(n+1)} \tag{26}$$

$$w_{r,0}'^i(n+1) = w_{r,0}^{i,L}(n+1)$$

$$w_{r,j}'^i(n+1) = \left(w_{r,j}^{i,L}(n+1) + w_{r,j}^{i,H}(n+1)\right)/2 \quad 1 \leq j \leq r_{\text{res}} - 2$$

$$w_{r,r_{\text{res}}-1}'^i(n+1) = w_{r,r_{\text{res}}-1}^{i,H}(n+1)$$

$$\forall_{i=0,1,\ldots r_{\text{res}}-1} \quad w_{r,j}^i(n+1) = \begin{cases} w_{r,j}'^i(n+1) & \text{if } \zeta > \zeta_i(n) \\ w_{r,j}^{i*}(n+1) & \text{if } \zeta \leq \zeta_i(n) \end{cases}.$$

The coefficients R_a^r and R_b^r determine a smoothing level and a diffusion speed, respectively. To increase numerical precision, the term p_j^{-1} is computed directly from the V_j^{i*} values, which is important because the values may get extremely low. Note the computation of two values $w_{r,j}^{i,L}(n+1)$ and $w_{r,j}^{i,R}(n+1)$, and then the taking of their mean, with the exception of special cases at the two LUT values having indexes 0 and $r_{\text{res}} - 1$. It is performed to maintain symmetry of the regularization of the LUT.

9

4.3   COMPUTING THE VISIT TABLE

Let us finally discuss computing the values in the visit table. Let the values $V_j^{i*}(n)$, that is the values of the visit table before a possible diffusion, have an equation as follows

$$V_j^{i*}(0) = V_p$$

$$V_j^{i*}(n+1) = \begin{cases} l\Big((1 - R_c^r)V_j^i(n)\Big) & \text{if } \Big(j \neq \lfloor S_i(n+1) \rfloor\Big) \wedge \\ & \wedge \Big(j \neq \lceil S_i(n+1) \rceil\Big) \\[2ex] l\Big((1 - R_c^r)V_j^i(n)\Big)\Big(1+ \\ \quad +R_c^r\Big(\lfloor S_i(n+1) \rfloor + 1 - & \text{if } j = \lfloor S_i(n+1) \rfloor \\ \quad -S_i(n+1)\Big)\Big(1 - V_j^i(n)\Big)\Big) \\[2ex] l\Big((1 - R_c^r)V_j^i(n)\Big)\Big(1+ \\ \quad +R_c^r\Big(S_i(n+1) - & \text{if } j = \lceil S_i(n+1) \rceil \\ \quad -\lfloor S_i(n+1) \rfloor\Big)\Big(1 - V_j^i(n)\Big)\Big) \end{cases} \quad , \quad (27)$$

$$l(x) = \max\left(x, V_{\min}\right)$$

$$S_i(n+1) = \frac{I_i(n) - I_{\min}}{I_{\max} - I_{\min}}\left(r_{\text{res}} - 1\right)$$

$$j = 0 \ldots r_{\text{res}} - 1$$

where $S_i(n+1)$ scales $I_i(n)$ like in (7). The coefficient V_p is the initial value. The coefficient $V_{\min}$ has a very small positive value and is used because of the limited precision of the representation of real numbers, used in computers. The coefficient R_c^r determines how large is the loss of importance of the less recent iterations in computing of the values of the visit table.

The values obtained from (27) are used in diffusing the weight function LUT, as it was shown in (26), yet the visit table is also diffused, because of the reasons already discussed in Sec. 4.2. The visit table is diffused like the weight function LUT, but without smoothing – it was decided to omit the smoothing, because, in contrast to the weight function, the visit table does not directly affect the propagation of signals. The new values of the visit function are computed on basis of the values of the visit function from the previous iteration, and not from the current iteration, like it was in (26). Thus, there

is the following formula for diffusing the visit table:

$$V_j^{i,L}(n+1) = m_j - \frac{\tanh(R_a^r d_j)}{2R_a^r(1+R_b^r p_j)} \quad 0 \le j \le r_{\text{res}} - 2$$

$$V_{j+1}^{i,H}(n+1) = m_j + \frac{\tanh(R_a^r d_j)}{2R_a^r(1+R_b^r p_j^{-1})} \quad 0 \le j \le r_{\text{res}} - 2$$

$$d_j = V_{j+1}^{i*}(n+1) - V_j^{i*}(n+1)$$

$$m_j = \left(V_j^{i*}(n+1) + V_{j+1}^{i*}(n+1)\right)/2$$

$$p_j = \frac{V_{j+1}^{i*}(n+1)}{V_j^{i*}(n+1)} \tag{28}$$

$$V_0'^i(n+1) = V_0^{i,L}(n+1)$$

$$V_j'^i(n+1) = \left(V_j^{i,L}(n+1) + V_j^{i,H}(n+1)\right)/2 \quad 1 \le j \le r_{\text{res}} - 2$$

$$V_{r_{\text{res}}-1}'^i(n+1) = V_{r_{\text{res}}-1}^{i,H}(n+1)$$

$$\forall_{i=0,1,\dots r_{\text{res}}-1} \quad V_{r,j}^i(n+1) = \begin{cases} V_j'^i(n+1) & \text{if } \zeta > \zeta_i(n) \\ V_j^{i*}(n+1) & \text{if } \zeta \le \zeta_i(n) \end{cases}.$$

5 TESTS

In this section, tests of the presented networks will be presented.

Unless otherwise stated, the following coefficients are used in all of the presented tests: $\mu = 0.02$, $\nu = 2.5$, $r_{\text{res}} = 64$, $I_{\min} = -1$, $I_{\max} = 1$, $a_l = 0.15$, $a_h = 0.35$, $a_m = 1.1$, $\zeta = 0.05$, $R_a^r = 1 \cdot 10^{-4}$, $R_b^r = 1 \cdot 10^{-4}$, $V_p = 0.1$, $V_{\min} = 1 \cdot 10^{-16}$, $R_c^r = 0.001$. For the classic neural networks with linear weight functions only, $R_s(w, \Delta w)$ is linear because $R_a^s = 0$, and the weight decay coefficient R_b^s is equal to $2 \cdot 10^{-7}$. For the networks with diffused weight functions $R_s(w, \Delta w)$ is nonlinear and regularizes weight change at $R_a^s = 1$, but the weight decay is weaker instead – the coefficient R_b^s is equal to $1 \cdot 10^{-9}$.

Let in the name of a neural network there be the 'LW' prefix in the case of the classic feedforward networks with linear weight functions only and the 'NLW' prefix in the case of the introduced networks. Let after the prefix there be the number of nodes in the subsequent layers, since the input layer. Thus, a classic neural network named LW 2–4–4–1 would have the number of nodes in the input layer equal to 2, then there would be two layers with 4 nodes in each one, and finally there would be a single node in the output layer.

5.1 TWO SPIRALS

Because the sets tested in this section are relatively simple for the introduced networks, the resolution of the LUT tables was decreased to $r_{\text{res}} = 16$ and the diffusion was 'strengthened' by using relatively large R_b^r values.

Let us first test the generalization of the 'two spirals' set, one of the standard benchmarks for learning machines (Lang and Witbrock, 1988). The set, after centering around the point $(0,0)$, is shown in Fig. 3(a). Each observation in the set has three attributes (x_0, x_1, y), the first two are the argument attributes and the last one is the value attribute.
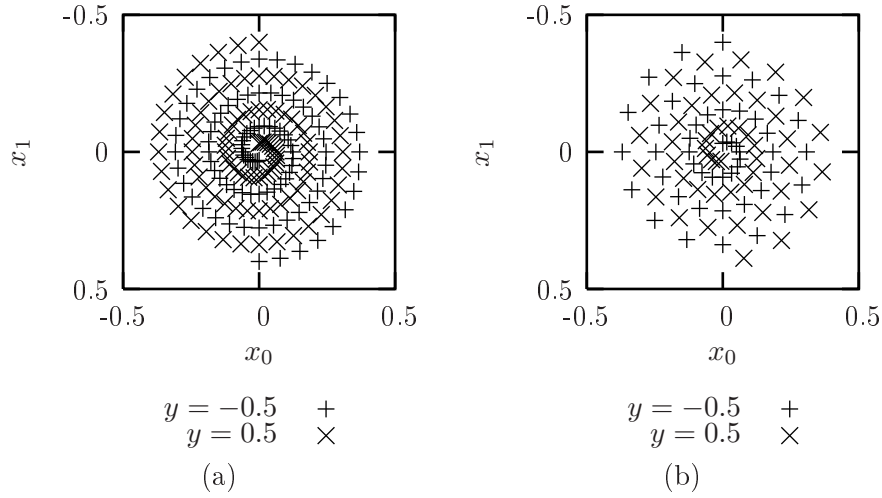
Figure 3: The (a) 'two spirals' and (b) 'two spirals sparse' data sets.

Even that the spirals may be regarded as relatively well defined because of the density of the observations determining them, this set is known to be very hard two–class problem to learn by the classic feedforward neural networks using the error backpropagation family of learning algorithms. Lang and Witbrock (1988) reported that the task could not be solved with the tested classic feedforward networks with connections only between the neighboring layers. To classify each observation in the training set, they developed a special architecture where each node was connected to all nodes in the subsequent layers and trained the network using error backpropagation with momentum. There were several trials of improving the learning algorithms to train neural networks more efficiently. For example Fahlman and Lebiere (1990) developed a learning algorithm that grew the trained neural network by adding new trained units to the network. They successfully applied the algorithm to the problem of the two spirals, but even that the trained neural network learned to classify all observations in the training set, the generalization quality was relatively poor – the decision border was very rough and it even crossed the arms of the spirals in some places.

The functions of the trained networks will be sampled, and presented as two–dimensional gray scale raster images of the size 128×128. The upper left corner pixel is at $(-0.5, -0.5)$ and the lower right corner pixel is at $(0.5, 0.5)$. Black pixels on the images denote -0.5 and white ones denote 0.5, and there is a gray scale between the two extreme values. The output values of the neural network that are less than -0.5 or greater than 0.5 will be shown also as black or white pixels, respectively.

In Fig. 4 results for some networks are shown. It is seen that the two presented LW networks could not even classify the training set within $1 \cdot 10^6$ iterations. The NLW network with the diffusion speed coefficient $R_b^r = 1 \cdot 10^{-4}$ generalized the training set with a somewhat rough decision border. Increasing the diffusion speed by increasing the value of the diffusion speed coefficient R_b^r to 0.01 caused that the generalization was much better.

Let us now discuss another training set, derived from the previous one. Let the observations within each of the spiral arms would be counted from the inner beginning of each of the arms. The set is created by removing each odd observation in one of the spiral arms and each even observation in the other of the arms. Let the set be called 'two spirals sparse'. The set is shown in Fig. 3(b). Such a way of removing the observations
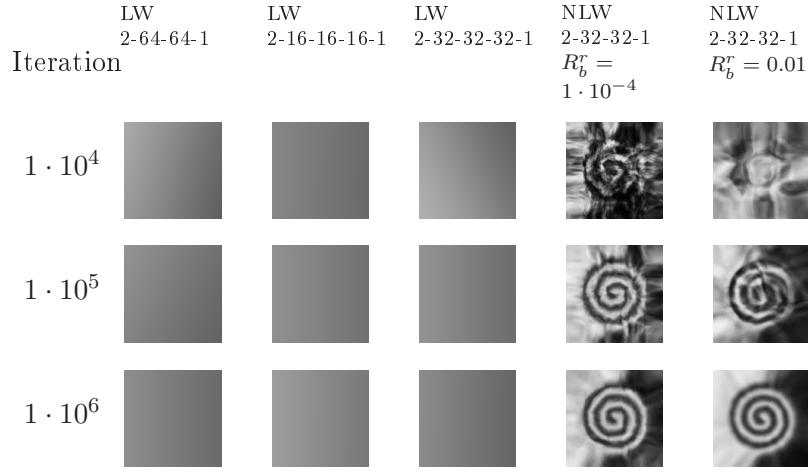
12

Iteration	LW 2-64-64-1	LW 2-16-16-16-1	LW 2-32-32-32-1	NLW 2-32-32-1 $R_b^r = 1 \cdot 10^{-4}$	NLW 2-32-32-1 $R_b^r = 0.01$
$1 \cdot 10^4$					
$1 \cdot 10^5$					
$1 \cdot 10^6$					

Figure 4: Output images for the 'two spirals' set.

was used to get a special type of patterns in the set. The patterns create two families, 'along arms' and 'radial', as it is illustrated in Fig. 5. It can be told that in the inner
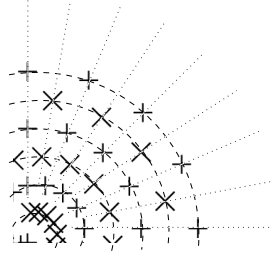
Figure 5: Two families of patterns in the 'two spirals sparse' data set, denotes with different types of lines.

side of the spirals the 'along arms' pattern is more strong than the 'radial' one, because of the relation between appropriate distances between observations. Conversely, the 'radial' pattern is stronger in the outer regions of the arms of the spirals. The third type of pattern is created by the halves of the outer turns of each of the spirals. Because the halves are not covered from the outside by any observations, the value attributes of the observations creating the halves may possibly be extrapolated to the outside, so that outside the spirals the generalizing function may roughly have values greater than 0 for $x_1 > 0$ and less than 0 for $x_1 < 0$. Let the task be to generalize the discussed set so that the strengths of the two families of patterns and of the third discussed pattern would be appropriately reflected in the generalizing function. The gradual transition of patterns in the discussed set will allow for testing the evenness of generalizing different regions of the space of argument attributes of the observations.

In Fig. 6 results for networks having some different architectures are shown. It can be seen that the presented LW networks did not classify the training set within $1 \cdot 10^6$ iterations. The NLW network with diffusion speed coefficient $R_b^r = 1 \cdot 10^{-4}$ generalized the training set but with rather severe problems. After increasing values of the diffusion speed coefficient to $R_b^r = 0.01$ the generalization was much better – all three discussed

types of patterns are seen. Further increase of the diffusion speed coefficient R_b^r to 0.1 rather did not give any improvements – as it can be seen in the output images, the learning process slowed down and in the particular run, at the $1 \cdot 10^6$th iteration, the generalizing functions seemed to be relatively less 'even'. It can thus be seen that for the

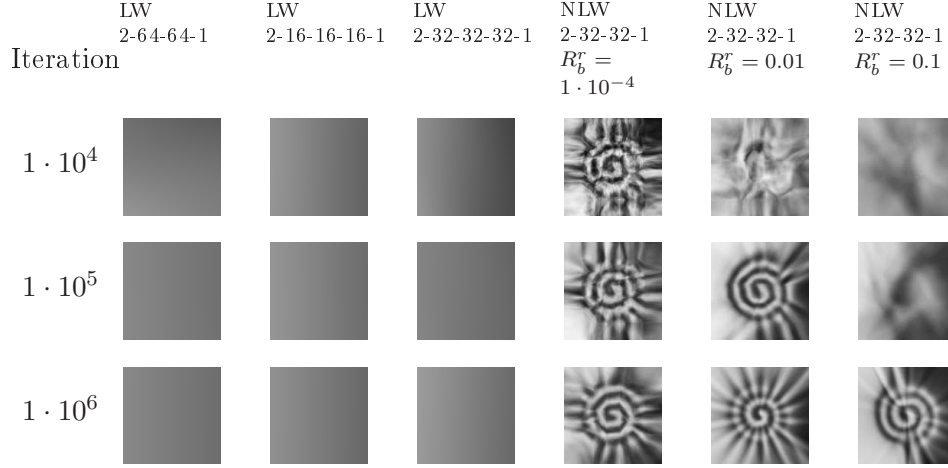| Iteration	LW 2-64-64-1	LW 2-16-16-16-1	LW 2-32-32-32-1	NLW 2-32-32-1 $R_b^r = 1 \cdot 10^{-4}$	NLW 2-32-32-1 $R_b^r = 0.01$	NLW 2-32-32-1 $R_b^r = 0.1$
$1 \cdot 10^4$						
$1 \cdot 10^5$						
$1 \cdot 10^6$						

Figure 6: Output images for the 'two spirals sparse' set.

discussed training set the coefficient $R_b^r = 0.01$ seems to be the most appropriate from the tested ones.

5.2 LEARNING A HIGH DIMENSION SET

One of the reasons of using heuristic nonlinear regressors like the feedforward neural networks can be their ability to scale well with the number of dimensions of the generalized set. In this section, an example of learning a set, that has not only a high dimension but is also relatively complex, is presented. Standard values of coefficients were used for training the neural networks with this complicated set.

Because the author did not found a standard benchmark of the required complexity, a custom training and test sets were used. Let the generating function of the sets be as follows:

$$
\begin{aligned}
y\left(x_0, x_1, x_2, x_3, x_4\right) = {} & \sin(4x_0)\cos(2x_1 + 3x_2) \\
& \left(\left(\frac{\sin(10x_2 + 10x_3) + 1}{2}\right)^{1/2} - 1/2\right) \\
& \sin(x_3 - 4x_1 x_4) \\
& \left(\left(\frac{\sin(10x_0 - 10x_2 + 10x_3) + 1}{2}\right)^{1/2} - 1/2\right) \\
& \cos(5x_1 x_2 x_4).
\end{aligned}
\tag{29}
$$

Using this equation, tuples $(x_0, x_1, x_2, x_3, x_4, y)$ were generated, where x_i were random values

$$
x_i = \mathrm{rand}() - 0.5 \quad i = 0, 1, \ldots 4,
\tag{30}
$$

where random() is a uniform random number generator, generating values in the range $\langle 0, 1 \rangle$. $1.8 \cdot 10^6$ such tuples were used in the training set, and independently generated $2 \cdot 10^5$ tuples were used in the test set.

Such relatively complex generalized sets like the discussed one can require substantial training times. Let us estimate the times for the LW and NLW networks. In the used

14

implementation, the iteration time/number of connections ratio was found to be finely generalized by linear functions, as follows:

- For the LW networks:

  - training time

  $$\mathrm{ll}(n) = -0.81 + 0.013n \tag{31}$$

  - propagation only time

  $$\mathrm{lp}(n) = -0.28 + 0.004n \tag{32}$$

- For the NLW networks:

  - training time

  $$\mathrm{nl}(n) = -1.84 + 0.056n \tag{33}$$

  - propagation only time

  $$\mathrm{np}(n) = -0.39 + 0.010n \tag{34}$$

where n is the number of connections and time is in milliseconds. In Fig. 7 a diagram of mean square error, further denoted by MSE, against the estimated times is shown for several LW and NLW networks, trained with the set 'dim6-2'. It is seen that the

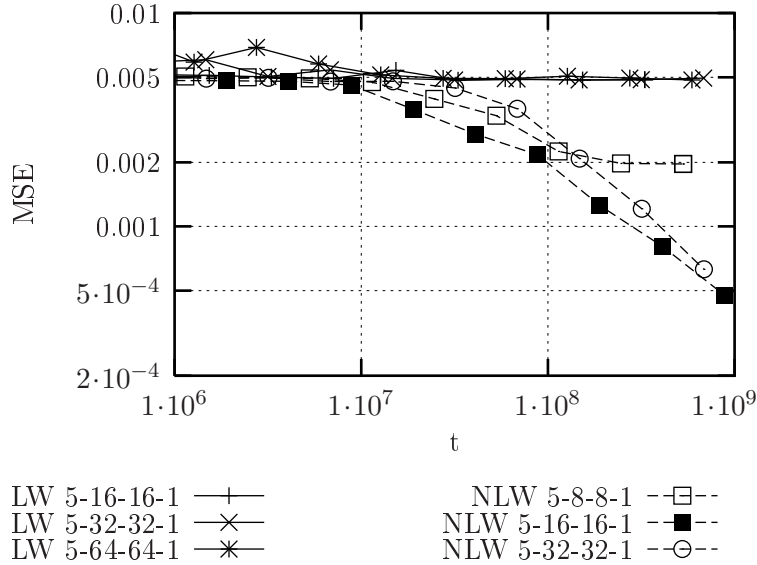| LW 5-16-16-1 | ⎯⎯+⎯⎯ | NLW 5-8-8-1 | --☐-- |
| LW 5-32-32-1 | ⎯⎯✕⎯⎯ | NLW 5-16-16-1 | --■-- |
| LW 5-64-64-1 | ⎯⎯✳⎯⎯ | NLW 5-32-32-1 | --○-- |

Figure 7: MSE diagram for different neural networks trained with the 'dim6-2' set, against estimated time.

NLW networks reach MSE even about ten times lower after similar training times in comparison to the LW networks.

6 CONCLUSIONS

The introduced neural networks have the ability of generalization of relatively complex sets in a relatively short time. The diffused weight functions can be adapted to be used in other architectures than the feedforward one, for example in the recurrent ones. The

weight functions can also be used in the spiking neural networks – for example, the number of spikes in a given period of time or phase differences between spike trains in some inputs of a neuron may be arguments of a diffused weight function, and the value of the function in turn could determine the strength of a connection.

References

Abbott, L. F., Varela, J. A., Sen, K., and Nelson, S. B. (1997). Synaptic depression and cortical gain control. *Science*, 275:220–224.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, New York.

Chen, C. T. and Chang, W. D. (1996). A feedforward neural network with function shape autotuning. *Neural Networks*, 9:627–641.

Fahlman, S. E. and Lebiere, C. (1990). The cascade–correlation learning architecture. In *Advances in Neural Information Processing Systems (D. S. Touretzky, Ed.)*, 2, pages 524–532, San Mateo. Morgan Kaufmann.

Greville, T. N. E. (1969). *Theory and Applications of Spline Functions*. Academic Press, New York.

Grossberg, S. (1969). On the production and release of chemical transmitters and related topics in cellular control. *J. Theor. Biol.*, 22:325–364.

Guarnieri, S. and Piazza, F. (1999). Multilayer feedforward networks with adaptive spline activation function. *IEEE Trans. Neural Networks*, 10(2):672–683.

Gurney, K. (1992a). Training nets of harware realisable sigma–pi units. *Neural Networks*, 5:289–303.

Gurney, K. (1992b). Weighted nodes and ram–nets: A unified approach. *Journal of Intelligent Systems*, 2:155–186.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA.

Koch, C. and Poggio, T. (1992). Multiplying with synapses and neurons. In T, T. M., Davis, J., and Zornetzers, editors, *Single Neuron Computation*, pages 315–345.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems*, volume 4, pages 950–957. Morgan Kaufmann Publishers, Inc.

Lang, K. and Witbrock, M. (1988). Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann Publishers, Inc.

Lorenzo, V., Piazza, F., and Uncini, A. (1998). Learning and approximation capabilities of adaptive spline activation neural networks. *Neural Networks*, 11:259–270.

McCulloch, W. S. and Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Piazza, D., Uncini, A., and Zenobi, M. (1993). Neural networks with digital lut activation function. In *Proceedings of IJCNN*, pages II–1401–1404, Nagoya, Japan.

Thomson, A. M. and Deuchars, J. (1994). Temporal and spatial properties of local circuits in neocortex. *Trends Neurosci*, 17:119–126.

Trentin, E. (2001). Networks with trainable amplitude of activation functions. *Neural Networks*, 14:471–493.

Uncini, A., Capparelli, F., and Piazza, F. (1998). Fast complex adaptive spline neural networks for digital signal processing. In *Proc. of IJCNN*, pages 903–909, Anchorage, Alaska.

Wang, J. N., Lay, S. R., Maecler, M., Martin, R. D., and Schimert, J. (1994). Regression modeling in back–propagation projection pursuit learning. *IEEE Trans. on Neural Networks*, 3(5):342–353.