

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

Technical Report IITiS-2003-02-23-1-1.06

Abstract

In this paper, feedforward neural networks are presented that have nonlinear weight functions based on look-up tables, that are specially smoothed in a regularization called the diffusion. The idea of such a type of networks is based on the hypothesis that the greater number of adaptive parameters per a weight function might reduce the total number of the weight functions needed to solve a given problem. Then, if the computational complexity of a propagation through a single such a weight function would be kept low, then the introduced neural networks might possibly be relatively fast.

A number of tests is performed, showing that the presented neural networks may indeed perform better in some cases than the classic neural networks and a number of other learning machines.

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

1 INTRODUCTION

Introducing adaptive nonlinear processing into the weight functions of feedforward, densely connected neural network gives the 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 the respective N adaptive nonlinear processing units in the classic neural networks.

The feedforward neural networks presented in this paper have nonlinear weight functions based on look-up tables – a look-up table represents nodes of a piecewise linear interpolation of the arguments of a weight function. Thanks to this, only one or two nodes in the table need to be read to propagate a signal. This causes that only a chosen subset of the parameters is used during a propagation of a given signal, what can make the propagation time reasonable despite a very large number of adaptive parameters. Another quality of the described neural networks is that the subsequent parameters in the look-up table control propagation of subsequent ranges in the domain of the weight function. Because of this, it can be hypothesized that two subsequent adaptive parameters in the look-up table are with some propability likely to control close points in the input space of the neural network. An advantage from the property can be taken during additional regularization of the weight function during the training process. Such a regularization is very important in the case of the presented neural networks, because it reduces the problems with the lack of smoothness of the look-up table based adaptive functions, that may cause serious generalization problems as reported by Piazza et al. [1993].

The regularization, called *diffusion*, works as follows. During the training process, the values of the propagated signals are traced to build ‘visit’ tables related to the frequency of falling of the look-up table arguments into different ranges of values. Each nonlinear weight function has such an accompanying visit table. On the basis of the tables, a regularization is performed, which ‘diffuses’ values within the look-up tables from the more ‘visited’ regions of the look-up tables to the less ‘visited’ regions of the look-up tables. This way, values of signals are extrapolated within a weight function, and in effect it might be likely that the values representing the generalized function are extrapolated by the discussed neural networks as well.

Schematic examples of generalization in the cases of a linear, spline-like and LUT weight functions are illustrated in Fig. 1. The lack of smoothness of the weight function

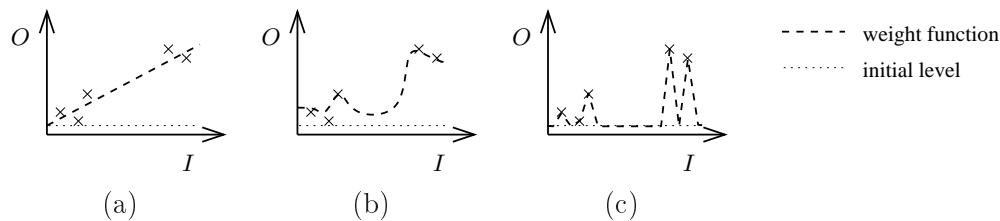


Figure 1: A schematic example of generalization using (a) linear, (b) smooth and (c) LUT weight function types. I are arguments and O are values of the weight functions, and crosses schematically denote points that minimize the training error function.

in (c) may clearly decrease the generalization quality Piazza et al. [1993]. Some adaptive parameters are completely independent from the training samples in that case. The weight function (b) is smooth and has a relatively small number of adaptable parameters to improve generalization – networks with similar, spline-like adaptive activation functions have been presented by Uncini, Capparelli and Piazza Uncini et al. [1998], Vecchi, Piazza and Uncini Vecchi et al. [1998], and Guarnieri and Piazza Guarnieri and Piazza [1999]. Such a lower number of adaptive parameters, however, is exactly what we want to elude in the proposed architecture. An example of generalization using a weight function with a large number of adaptable parameters, like in (c), without and with the diffusion of the function, is shown in Fig. 7(a) and Fig. 9(a), respectively. It can be seen, comparing these figures and the respective generalizing functions, that the diffusion may clearly improve the generalization.

The use of high number of adaptable parameters for each connection may raise the question about bounds on the generalization performance of a learning machine Vapnik [1995a]. Yet, for first it should be noted, that the high number of the parameters per connection might result in a lower total number of connections needed. Secondly, even if the introduced neural networks might obviously have a very high Vapnik Chervonenkis dimension, so the risk bound would be very high, the VC dimension takes into account only the maximum number of training points that can be shattered by the learning machine. Thus it clearly can be a very ‘loose’ bound, in the sense that the other qualities of the learning machine can make the actual risk much lower than the bound. For example, the discussed diffusion can make the subsequent adaptable parameters in the LUT weight function dependent on each other, yet in computing of the VC dimension it is not taken into account at all.

2 NETWORKS WITH DIFFUSED WEIGHT FUNCTIONS

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. This way of describing a neuron is illustrated in Fig. 2

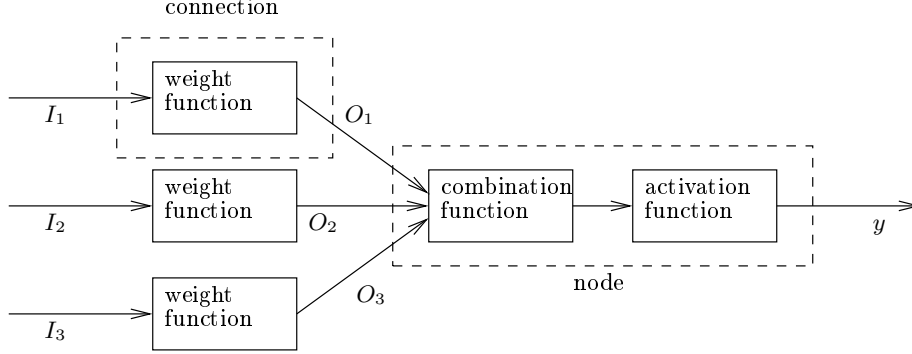


Figure 2: Structure of an example neuron.

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 neurons McCulloch and Pitts [1943]

$$u_k^c(n) = \sum_{i \in M} O_i(n), \quad (1)$$

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

$$y_k(n) = u_k^a(u_k^c(n)) = \tanh(u_k^c(n)), \quad (2)$$

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

$$O_i(n) = w_s^i I_i(n), \quad (3)$$

where w_s^i is the connection scalar weight.

A LUT connection i has the following weight function

$$O_i(n) = w_l^i(n) I_i(n) + r(\mathbf{w}_r^i(n), I_i(n)), \quad (4)$$

where $w_l^i(n) I_i(n)$ 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 $j = 0 \dots r_{\text{res}} - 1$ and $w_{r,j}^i(n)$ is a j th element of $\mathbf{w}_r^i(n)$. The first coordinate of the points on the curve denotes arguments and the second one values of the function. The values I^j are equally distributed values as follows

$$I^j = I_{\min} + \frac{j}{r_{\text{res}} - 1}(I_{\max} - I_{\min}). \quad (5)$$

Let us 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 $r(\mathbf{w}_r^i(n), I(n))$ be computed using the following piece-wise linear interpolation:

$$\begin{aligned} r(\mathbf{w}_r^i(n), I(n)) &= ([S(n)] + 1 - S(n))w_{r,[S(n)]}^i(n) + (S(n) - [S(n)])w_{r,[S(n)]+1}^i(n) \\ S(n) &= \frac{I(n) - I_{\min}}{I_{\max} - I_{\min}}(r_{\text{res}} - 1). \end{aligned} \quad (6)$$

Alternatively, of course, another interpolation type, for example cubic spline interpolation Greville [1969], de Boor [1978], could be used.

The memory overhead of the presented networks grows linearly with r_{res} – each non-linear weight function needs only one additional table for the diffusion process discussed in Sec. 3.2.2, and the additional table is of the size of r_{res} . Thus, even with networks having thousands of connections, and r_{res} being of the order of hundreds, the overhead can be low on modern computers, because such computers often feature hundreds of megabytes of memory.

Because the piecewise linear interpolation requires only one or two adaptive parameters, the ratio of the number of the used parameters within a single propagation to the number of all of the parameters is less than or equal to $2/r_{\text{res}}$. Let us call the quality of using only some chosen adaptive parameters the selective parameters property. The property can make propagation very fast, while still retaining a large number of adaptive parameters available to the training process. The relatively fast propagation in the presented networks will be demonstrated in tests. If, instead of the piecewise interpolation, a function like a single polynomial would be used for a weight functions, then the property of selective parameters would obviously not apply, because each adaptive parameter would be needed to find a value of the function.

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 linear and the LUT components may in some cases perform substantially better than a network with only the LUT 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. A sample such a NN is illustrated in Fig. 3.

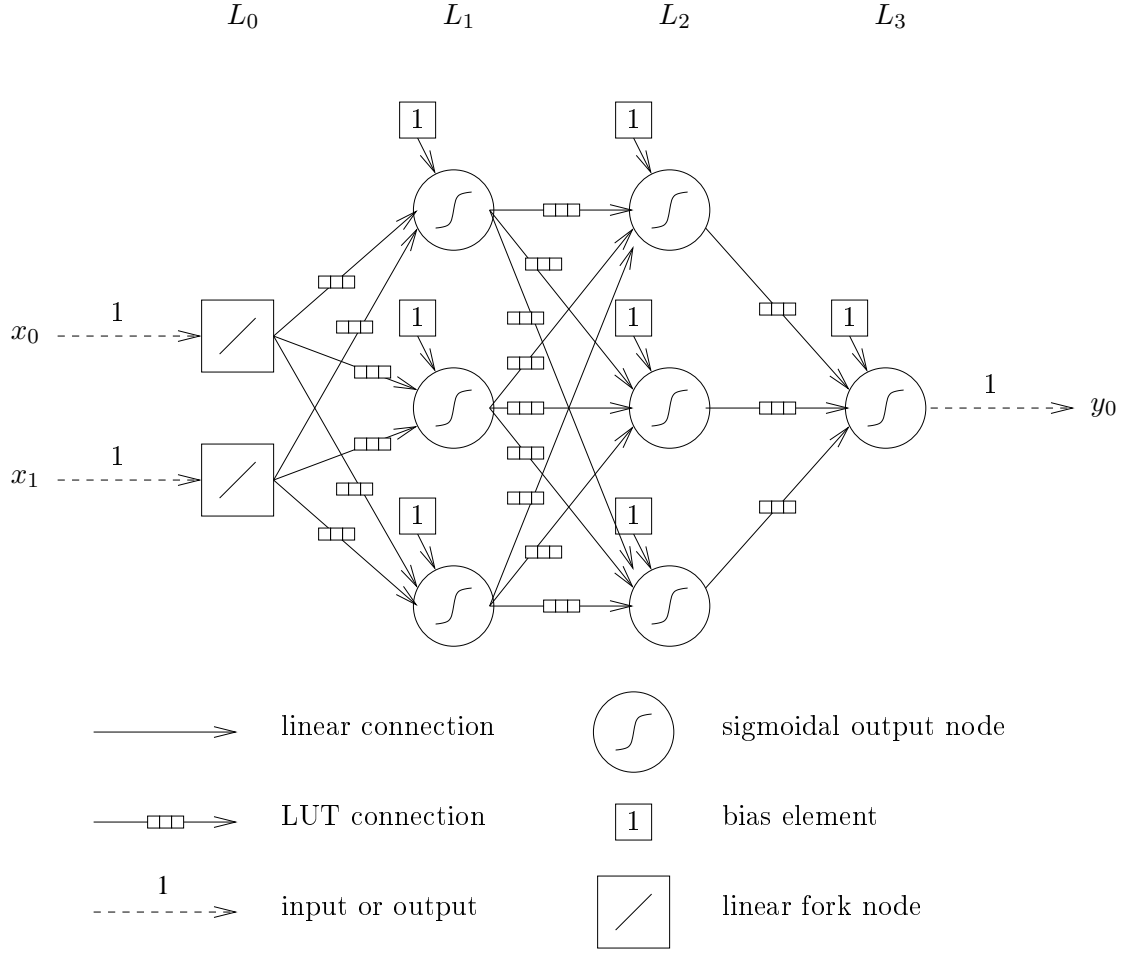


Figure 3: An example of a fully connected multilayer feedforward NN with nonlinear weight functions, L_i denotes the i th layer.

3 THE LEARNING ALGORITHM

The presented NNs have their distinct learning algorithm, consisting of a modified error backpropagation Rumelhart and McClelland [1996], Bishop [1995], Hertz et al. [1991] and a regularization of the weight functions.

A schematic diagram of a single iteration of the learning algorithm is presented in Fig. 4. In the beginning of an iteration, attributes of the training samples are propagated through the network. The propagation needs derivatives of the weight functions. In the case of the LUT weight function, approximation of the derivative is computed instead. During the propagation, weight functions are adapted and the visit functions are updated. Then, regularization is performed. Within the regularization, weight decay is performed on both the linear and nonlinear weight functions. Also, the nonlinear weight functions and the visit functions are diffused, according to the values in the visit functions.

Note that the order of the mentioned operations is not critical – the training process may have many iterations, and so the progressive changes within a single training iteration are usually relatively low. In the detailed description of the learning algorithm later in this section, for mathematical completeness, the blocks presented in Fig. 4 are tied in a

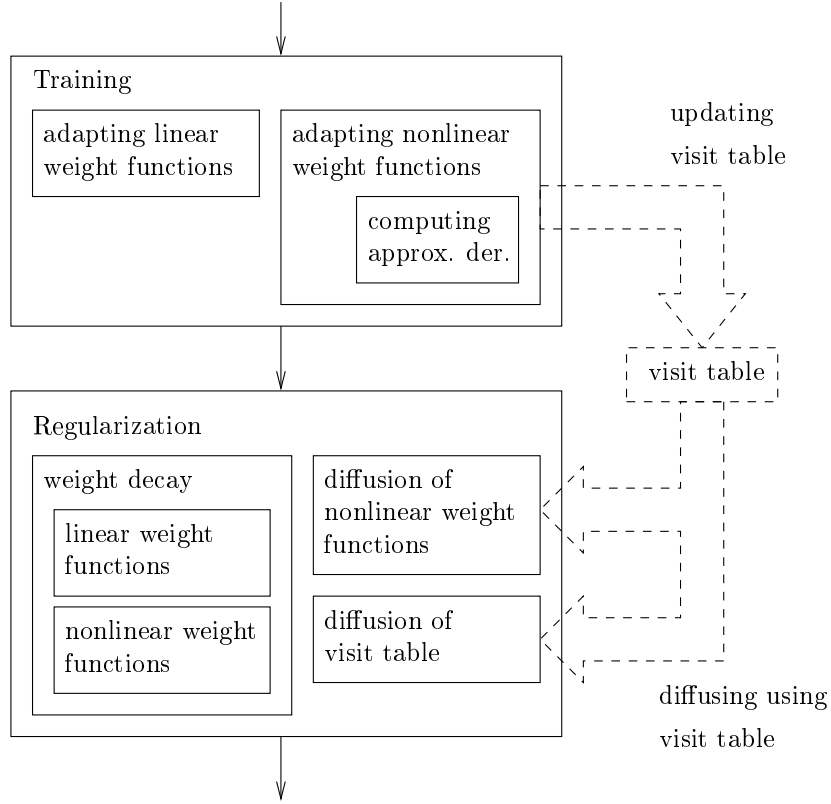


Figure 4: Diagram of a single iteration of the learning algorithm.

specific order, but the order is practically unimportant.

3.1 TRAINING

On-line training with error backpropagation Rumelhart and McClelland [1996], Bishop [1995], Hertz et al. [1991] is used.

Let a training set be given. Each sample k in the set has i argument attributes and j value attributes $\{x_0^k, x_1^k, \dots, x_{i-1}^k, d_0^k, d_1^k, \dots, d_{j-1}^k\}$ 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.

To keep the descriptions of the training algorithm and the regularization algorithm separate, here the regularization functions $R_d(w)$ and $R_s(w, \Delta w)$ will only be briefly mentioned, and described in detail in Sec. 3.2. 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 $(0, 1)$. 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

$$R_s(w, \Delta w) = \Delta w. \quad (7)$$

The regularization of the LUT component can be much more computationally complex than 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:

$$\zeta > \zeta_i(n), \quad (8)$$

where

$$\zeta_i(n) = \text{rand}(0.0, 1.0), \quad (9)$$

and the function $\text{rand}(0, 1)$ is a uniform random number generator which returns a random value within $\langle 0, 1 \rangle$. The more the ζ coefficient is lower than 1, the less is the mean computation complexity per training iteration, but the quality of the regularization may be worse. The LUT regularization can accordingly be ‘stronger’ to counterbalance its exclusion in some iterations. The random exclusion (8), instead of a regular one, is used to rule out possible resonance with the training samples. Condition (8) will be repeated in some equations in the later sections.

3.1.1 ADAPTING WEIGHTS OF THE LINEAR CONNECTIONS

The weights are randomly initialized before training. with values in the range $\langle -0.5, 0.5 \rangle$.

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:

$$w_s^i(n+1) = R_d(w_s^i(n) + \Delta w_s^i(n)), \quad (10)$$

where

$$\Delta w_s^i(n) = -R_s(w_s^i(n), \mu e_i(n) I_i(n)). \quad (11)$$

3.1.2 ADAPTING WEIGHTS OF THE LUT CONNECTIONS

A LUT component is randomly initialized before the training of the neural network with small values in the range $\langle -0.5, 0.5 \rangle$ and a constant 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:

$$w_l^i(n+1) = R_d(w_l^i(n) + \nu \Delta w_l^i(n)), \quad (12)$$

where

$$\Delta w_l^i(n) = -R_s(w_l^i(n), \mu e_i(n) I_i(n)). \quad (13)$$

The coefficient ν specifies a relation between the adaptation speed of the linear component and 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:

$$r(\mathbf{w}_r^{i**}(n+1), I_i(n)) = r(\mathbf{w}_r^i(n), I_i(n)) + \Delta w_r^i(n) \\ r(\mathbf{w}_r^{i*}(n+1), I_i(n)) = \begin{cases} R_d(r(\mathbf{w}_r^{i**}(n), I_i(n))) & \text{if } \zeta > \zeta_i \\ r(\mathbf{w}_r^{i**}(n), I_i(n)) & \text{if } \zeta \leq \zeta_i \end{cases}, \quad (14)$$

where

$$\Delta w_r^i(n) = -R_s \left(r(\mathbf{w}_r^i(n), I_i(n)), \mu e_i(n) \right). \quad (15)$$

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 is the value $r(\mathbf{w}_r^i(n), I_i(n))$ itself, the term $e_i(n)$ is used in (15) instead of the term $e_i(n)I_i(n)$ as in (11). The symbol $*$ in (15) and later in this section denotes a value before the diffusion of the LUT component. The diffusion which is described later in Sec. 3.2.2.

The equation (14) 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 adaptation 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 (14),

$$w_{r,j}^{i**}(n+1) = w_{r,j}^i(n) + \Delta w_r^i(n). \quad (16)$$

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 the amounts $\Delta w_L^r(n)$ and $\Delta w_H^r(n)$, respectively,

$$\begin{aligned} w_{r,j}^{i**}(n+1) &= w_{r,j}^i(n) + \Delta w_L^r(n) \\ w_{r,j+1}^{i**}(n+1) &= w_{r,j+1}^i(n) + \Delta w_H^r(n) \end{aligned} \quad (17)$$

such that

$$\frac{\Delta w_H^r(n)}{\Delta w_L^r(n)} = \frac{I_i(n) - I^j(n)}{I^{j+1}(n) - I_i(n)}. \quad (18)$$

Therefore, one of the modified points, denoted 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 (14) and (18), $\Delta w_L^r(n)$ and $\Delta w_H^r(n)$ have the following form

$$\begin{aligned} \Delta w_L^r(n) &= \Delta w_r^i(n) \frac{r_L}{2r_L^2 - 2r_L + 1} \quad r_L = I_{j+1}^i(n) - I_i(n) \\ \Delta w_H^r(n) &= \Delta w_r^i(n) \frac{r_H}{2r_H^2 - 2r_H + 1} \quad r_H = I_i(n) - I_j^i(n) \end{aligned} \quad (19)$$

As can be seen, the quality of selective parameters apply also to the adapting of the parameters – only one or two parameters are modified during training within a single iterations, independently of the value of r_{res} .

3.1.3 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 be described as follows

$$\frac{\partial v(\mathbf{w}_i(n), I_i(n))}{\partial I_i(n)} = c_l^i(n) + c_r^i(n), \quad (20)$$

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,

$$c_l^i(n) = w_l^i(n). \quad (21)$$

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 weight changes and could cause numerical instability Guarnieri et al. [1999]. Instead, the approximated derivative of a LUT component r of a connection i is given by

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

$$A = \{a_l, a_m a_l, a_m^2 a_l, \dots, a_f\} \quad a_f \leq a_h \quad a_m a_f > a_h \quad . \quad (22)$$

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

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.

3.2 REGULARIZATION

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

3.2.1 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]. 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. 3.1. In this section, the functions will be described in more detail.

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

$$R_d(w) = (1 - R_b^s)w. \quad (23)$$

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

A weight decay like in (23) 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 of regularization can still be very important Krogh and Hertz [1992]. In the presented algorithm, the following solution is proposed. The weight decay (23), 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:

$$R_s(w, \Delta w) = \begin{cases} \frac{\exp(R_a^s w \Delta w) - 1}{R_a^s w} & \text{if } w \neq 0 \wedge R_a^s \neq 0 \\ \Delta w & \text{if } w = 0 \vee R_a^s = 0 \end{cases} \quad (24)$$

where R_a^s determines the level of the regularization. As 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 absolute values.

3.2.2 DIFFUSION OF THE LUT COMPONENT

A given training sample can, by its very presence, make the point that it represents in the input space of the regressor, and the surroundings of the point, more statistically significant or defined. The ‘spreading’ of the significance over the surroundings is used in methods like for example the nearest neighbor or cubic spline interpolation Greville [1969], de Boor [1978]. Assuming that the weight functions are not very ‘jagged’, it can be hypothesized that two subsequent adaptive parameters in the look-up table are likely to control close points in the input space of the neural network. The idea behind diffusion is based just on that. A sample has a ‘significance’ and while its attributes are propagated through the network, the ‘significance’ is marked in respective regions of the weight functions by the means of the accompanying visit function. 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, thus heuristically performing an interpolation by the ‘spreading’ of the significance of samples like in the mentioned nearest neighbor or cubic spline interpolations.

The algorithm of diffusion was constructed so as to have low memory requirements and low time complexity. For each LUT component it needs only one additional visit 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 Fick’s diffusion equation, and the ‘significance’ estimation is heuristic. The algorithm, however, keeps the time of a single iteration relatively short. This may be important if there are many training samples, and in effect many iterations to propagate the attributes of the samples are required.

In the process of diffusion, roughly speaking, the weighted density of occurrence of LUT component arguments, at different regions of the LUT component domain, is computed. In computing 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 the LUT 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 the mean of two such neighboring values of the weight function LUT before diffusion be m . After diffusion, the value that had higher corresponding value in a visit table moves less towards m than the other value. 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.

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.

During the diffusion process, the LUT functions are also ‘smoothed’ by decreasing the absolute differences between neighboring values in the LUTs, to reduce the problems caused by the lack of smoothness as reported in 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. 3.1. We want to smooth $r(\mathbf{w}_r^{i*}(n+1), I_i(n))$ by making the 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 $w_{r,j}^i(n)$, $j = 0 \dots r_{\text{res}} - 1$, 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:

$$\begin{aligned}
w_{r,j}^{i,L}(n+1) &= m_j - \frac{\tanh(R_a^r d_j)}{2R_a^r(1 + R_b^r p_j)} \quad 0 \leq j \leq r_{\text{res}} - 2 \\
w_{r,j+1}^{i,H}(n+1) &= m_j + \frac{\tanh(R_a^r d_j)}{2R_a^r(1 + R_b^r p_j^{-1})} \quad 0 \leq j \leq r_{\text{res}} - 2 \\
d_j &= w_{r,j+1}^{i*}(n+1) - w_{r,j}^{i*}(n+1) \\
m_j &= (w_{r,j}^{i*}(n+1) + w_{r,j+1}^{i*}(n+1))/2 \\
p_j &= \frac{V_{j+1}^{i*}(n+1)}{V_j^{i*}(n+1)} \\
w_{r,0}^i(n+1) &= w_{r,0}^{i,L}(n+1) \\
w_{r,j}^i(n+1) &= (w_{r,j}^{i,L}(n+1) + w_{r,j}^{i,H}(n+1))/2 \quad 1 \leq j \leq r_{\text{res}} - 2 \\
w_{r,r_{\text{res}}-1}^i(n+1) &= w_{r,r_{\text{res}}-1}^{i,H}(n+1) \\
\forall_{i=0,1,\dots,r_{\text{res}}-1} \quad w_{r,j}^i(n+1) &= \begin{cases} w_{r,j}^i(n+1) & \text{if } \zeta > \zeta_i(n) \\ w_{r,j}^{i*}(n+1) & \text{if } \zeta \leq \zeta_i(n) \end{cases} .
\end{aligned} \tag{25}$$

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 these values may get extremely low. The computation of the two values $w_{r,j}^{i,L}(n+1)$ and $w_{r,j}^{i,H}(n+1)$, and then computing of their mean, with the exception of special cases at the two LUT values having indexes 0 and $r_{\text{res}} - 1$, is performed to maintain symmetry of the regularization of the LUT.

3.2.3 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

$$\begin{aligned}
 & V_j^{i*}(0) = V_p \\
 & V_j^{i*}(n+1) = \begin{cases} l((1 - R_c^r)V_j^i(n)) & \text{if } (j \neq \lfloor S_i(n+1) \rfloor) \wedge \\ & \wedge (j \neq \lceil S_i(n+1) \rceil) \\ \\ l((1 - R_c^r)V_j^i(n)) \left(1 + \right. & \text{if } j = \lfloor S_i(n+1) \rfloor \\ \quad \left. + R_c^r (\lfloor S_i(n+1) \rfloor + 1 - \right. & \\ \quad \left. - S_i(n+1)) (1 - V_j^i(n)) \right) & \\ \\ l((1 - R_c^r)V_j^i(n)) \left(1 + \right. & \text{if } j = \lceil S_i(n+1) \rceil \\ \quad \left. + R_c^r (S_i(n+1) - \right. & \\ \quad \left. - \lceil S_i(n+1) \rceil) (1 - V_j^i(n)) \right) & \end{cases} \quad (26) \\
 & l(x) = \max(x, V_{\min}) \\
 & S_i(n+1) = \frac{I_i(n) - I_{\min}}{I_{\max} - I_{\min}} (r_{\text{res}} - 1) \\
 & j = 0 \dots r_{\text{res}} - 1 \quad .
 \end{aligned}$$

where $S_i(n+1)$ scales $I_i(n)$ like in (6). 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 (26) are used in diffusing the weight function LUT, as was shown in (25), yet the visit table is also diffused, because of the reasons already discussed in Sec. 3.2.2. The visit table is diffused like the weight function LUT, but without smoothing – it was decided to omit the smoothing here, because, in contrast to the weight function, the visit table does not directly affect the propagation of signals.

Thus, the following formula is used for diffusing the visit table:

$$\begin{aligned}
V_j^{i,L}(n+1) &= m_j - \frac{d_j}{2(1+R_b^r p_j)} \quad 0 \leq j \leq r_{\text{res}} - 2 \\
V_{j+1}^{i,H}(n+1) &= m_j + \frac{d_j}{2(1+R_b^r p_j^{-1})} \quad 0 \leq j \leq r_{\text{res}} - 2 \\
d_j &= V_{j+1}^{i*}(n+1) - V_j^{i*}(n+1) \\
m_j &= (V_j^{i*}(n+1) + V_{j+1}^{i*}(n+1))/2 \\
p_j &= \frac{V_{j+1}^{i*}(n+1)}{V_j^{i*}(n+1)} \\
V_0^i(n+1) &= V_0^{i,L}(n+1) \\
V_j^i(n+1) &= (V_j^{i,L}(n+1) + V_j^{i,H}(n+1))/2 \quad 1 \leq j \leq r_{\text{res}} - 2 \\
V_{r_{\text{res}}-1}^i(n+1) &= V_{r_{\text{res}}-1}^{i,H}(n+1) \\
\forall_{i=0,1,\dots,r_{\text{res}}-1} \quad V_{r,j}^i(n+1) &= \begin{cases} V_j^i(n+1) & \text{if } \zeta > \zeta_i(n) \\ V_j^{i*}(n+1) & \text{if } \zeta \leq \zeta_i(n) \end{cases} .
\end{aligned} \tag{27}$$

4 TESTS

In this section, the presented networks will be tested and compared to some other neural and non neural learning machines.

Unless otherwise stated, the following coefficients, selected in a number of preliminary trials, are used in the tests in this section: $\mu = 0.02$, $\nu = 2.5$, $r_{\text{res}} = 64$, $I_{\text{min}} = -1$, $I_{\text{max}} = 1$, $a_l = 0.15$, $a_h = 0.35$, $a_m = 1.1$, $\zeta = 0.05$, $R_a^r = 1 \cdot 10^{-4}$, $R_b^r = 1 \cdot 10^{-4}$, $V_p = 0.1$, $V_{\text{min}} = 1 \cdot 10^{-16}$, $R_c^r = 0.001$. For the classic neural networks with linear weight functions only, to make their weight decay like the classic one described in Krogh and Hertz [1992], $R_s(w, \Delta w)$ 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 $1 \cdot 10^{-9}$. To show the regularization capabilities of the presented networks, only the μ , r_{res} and R_b^r coefficients will actually be fitted to different generalized data sets, except for some special cases, and the rest of the coefficients will be constant.

The ‘LW’ prefix is used to represent the classic feedforward networks with linear weight functions only, whereas the ‘NLW’ prefix corresponds to the introduced networks. Following this prefix is the number of nodes in the subsequent layers, after the input layer. Thus, a classic neural network named LW 2–4–4–1 would have 2 nodes in the input layer, followed by two layers with 4 nodes in each, and finally a single node in the output layer.

4.1 TIME COMPLEXITY

The speed of a signal propagation through a connection and the time of modifying a connection weight can be substantially different for connections with the linear and the

$a + bn$	a	b
LW training	$-0.81^{+2.56}_{-1.59}$	0.013
LW propagation only	$-0.28^{+1.42}_{-0.58}$	0.004
NLW training	$-1.84^{+4.30}_{-2.07}$	0.056
NLW propagation only	$-0.39^{+1.20}_{-0.84}$	0.010

Table 1: Time complexity for a single iteration generalized using linear regression.

LUT weights. In this section, some time measurements are provided for estimation and comparison of the learning and propagation performance of both the classic and the introduced networks. The time results are for a particular implementation and should be interpreted with care, of course.

A number of architectures was tested, with different number of inputs, outputs, layers and nodes within a layer. The iteration time to the number of connections ratio was generalized by linear functions of the form $a + bn$, as shown in Table 1, where n is the number of connections, the time is in milliseconds and the $+/-$ values indicate the lower and upper parallel bounding lines of the measured times, respectively. The difference in the iteration only times is only of about 2.5 times, what results from using a fast piecewise linear interpolation, resulting in the selective parameters property, in the nonlinear weight functions.

The fitted function for learning times against r_{res} is as follows

$$\text{nrl}(r_{\text{res}}) = 38.1^{+310}_{-66.6} + 0.113r_{\text{res}}$$

For r_{res} increase of 16 times, from 16 to 256, there is a time increase per iteration of about 1.7. It is because of the selective parameters property of the weight functions, and because $\zeta = 0.05 \ll 1$ so r_{res} value is critical for time performance in only about 5% of all training iterations.

4.2 DIFFUSION OF LUT WEIGHTS

The training set generalized in this section, for visualization purposes, is a raster image. In each sample of the set there are three attributes (x, y, v) . The attributes x and y are the argument ones and represent coordinates in a two-dimensional mesh and the v attribute is the value one. The data set ‘circle’ is seen in Fig. 5(a). The image has the resolution of 64×64 . 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 are denoted by -0.5 and white ones by 0.5 , with a gray scale between the two values. The NN generalization function will be shown in a similar way, but the values less than -0.5 or greater than 0.5 will also be shown as black or white pixels, respectively. The mask in Fig. 5(b) shows by white pixels the respective samples that are chosen for the training.

Let us first test the introduced NNs without the diffusion of the weight functions, to compare it later with NNs that have the diffusion. To disable the weight functions diffusion, let $R_b^r = 0$. Let the LUT smoothing will also be disabled by setting $R_a^r = 0$ to

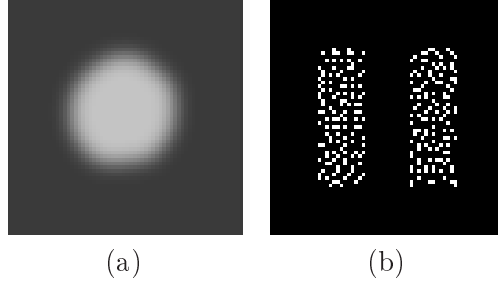


Figure 5: The ‘circle’ data set (a) and (b) its mask.

show the generalization similar to that seen in Fig.1(c). Let the ζ coefficient be equal to 1 in the examples in this section, as in the section not the time efficiency is tested, but the generalization ability is presented, and $\zeta = 1$ allows for smoother diagrams of weights. In Fig. 6 images of the generalizing function representing the approximated training set are shown for a 1–1 NN at some different iterations. The used training set leaves relatively large regions in the space of the argument attributes unknown by a trained NN. Because of the generalization ability the NN should ‘extrapolate’ learned samples over these regions. As can be seen in Fig. 6, the NN without diffusion generalizes relatively poorly at the $1 \cdot 10^5$ th iteration, showing problems resulting from the lack of both smoothness like it was described in Piazza et al. [1993]. In Fig. 7, diagrams showing the nonlinear weight functions and the visit tables at some iterations of the training process are presented. The intensity representing the values in visit tables is nonlinearly related to the values, to make the smaller ones better visible. The lack of diffusion of the LUT weight functions is clearly seen.

Let us then test an identical NN, but with the standard values of R_a^r and R_b^r of $1 \cdot 10^{-4}$. In Fig. 8 a much improved generalization can be seen after the first $1 \cdot 10^5$ iterations, in compare to that shown in Fig. 6. In Fig. 9 the diffusion of LUT weight functions can be seen.

Some tests with various values of the diffusion speed coefficient R_b^r will also be performed in Sec. 4.4.

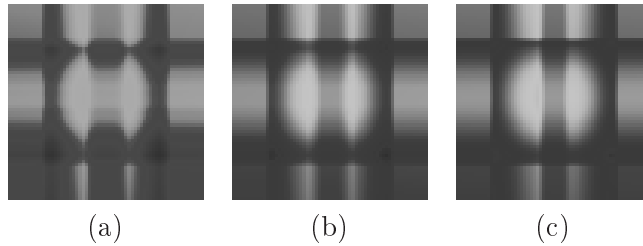


Figure 6: Images of the generalizing function after (a) 1000, (b) 10000 and (c) $1 \cdot 10^5$ iterations, respectively, of a 1–1 NN for an image ‘circle’, $R_a^r = 0$, $R_b^r = 0$.

4.3 SMALL SIZE DATA SETS

The type of neural networks presented in this paper were designed for generalization of sets of a very high complexity or highly nonlinear, for example needing hundreds of thou-

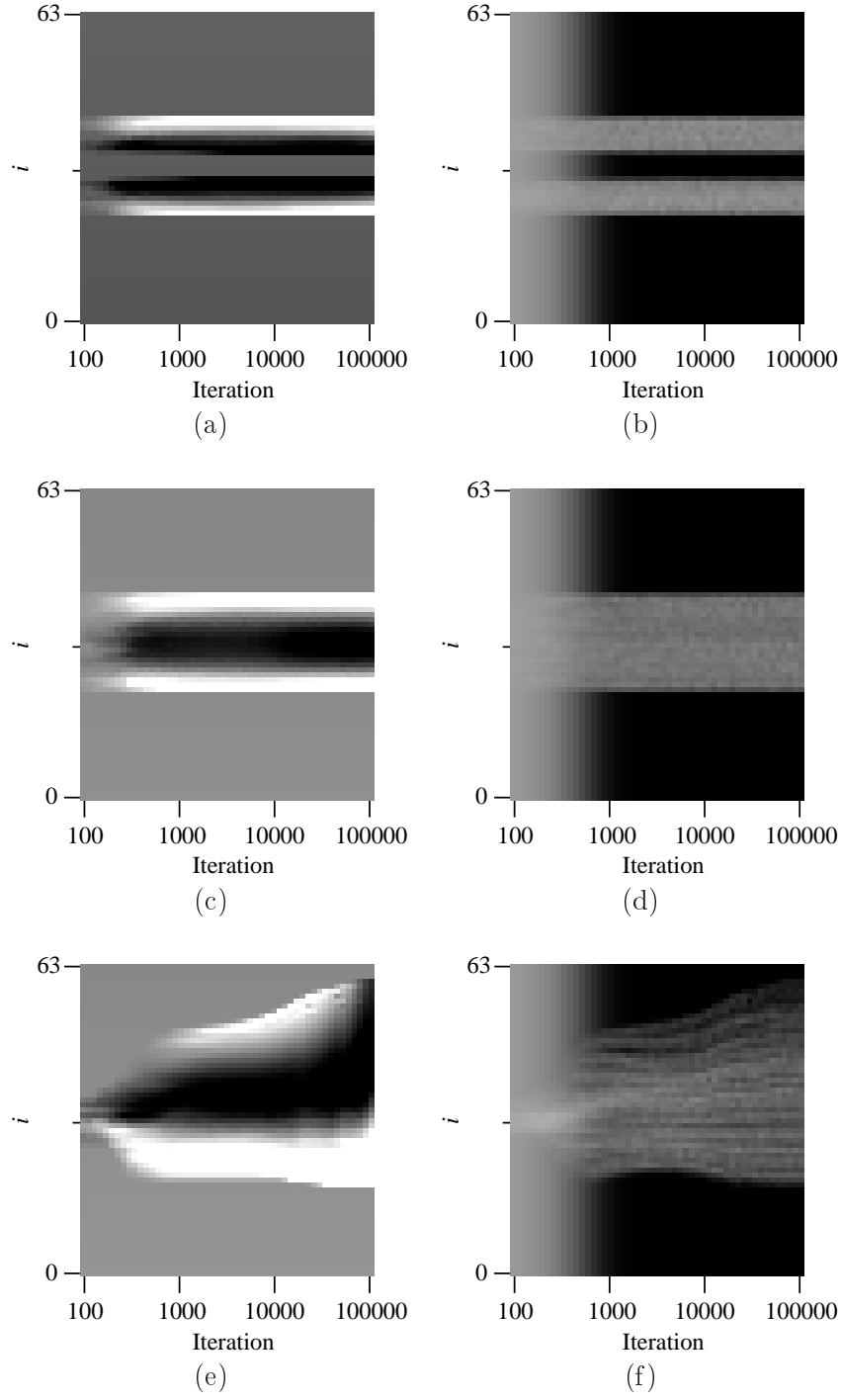


Figure 7: A LUT weights diagram of a 1-1 NN for an image ‘circle’, $R_a^r = 0$, $R_b^r = 0$. Connection $x_0 \rightarrow s(1,0)$ (a) LUT component, (b) visit table, connection $x_1 \rightarrow s(1,0)$ (c) LUT component, (d) visit table, connection $s(1,0) \rightarrow s(2,0)$ (e) LUT component, (f) visit table. The i axes denote weight function LUT (a)(c)(e) or visit table (b)(d)(f) indices, x_j denotes j th input of the NN and $s(l,n)$ denotes a node n in the l th layer.

sands of samples to be roughly represented, or that have patterns like the ‘two spirals’ set

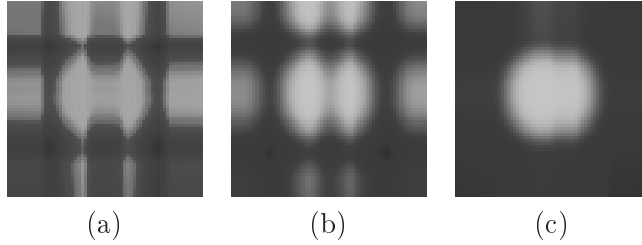


Figure 8: Images of the generalizing function after (a) 1000, (b) 10000 and (c) $1 \cdot 10^5$ iterations, respectively, of a 1-1 NN for an image ‘circle’, $R_a^r = 1 \cdot 10^{-4}$, $R_b^r = 1 \cdot 10^{-4}$.

Lang and Witbrock [1988], tested in the next section. Many generalized data sets, however, are much smaller and more linear. Yet, it still may be a difficult task to generalize them well – in Draghici [2001] performance comparison of several neural- and non-neural learning algorithms shows substantial variance in the percentage of properly classified test samples in the case of some data sets from the UCI repository Blake and Merz [1998], of which all have less than one thousand samples. It can be important for a learning machine to have a good performance on a wide range of data sets, so in this section performance comparison is performed on some relatively small data sets. In this test, classification results of classic neural networks with linear weight functions, several other neural and non-neural learning algorithms, and the introduced neural networks are compared. The generalized sets are from the mentioned UCI repository and generally have patterns of a moderate nonlinearity.

The coefficient R_b^r was set to 0.02 to increase the diffusion rate so to counterpart the relatively sparse samples in the generalized sets.

The generalized sets were randomly divided into training sets containing 80% of samples and test sets containing 20% of samples. 10 runs of each tested architecture of the classic LW networks, and of the introduced NLW networks, were performed, and the average values were shown. For comparison, the SVM Vapnik [1995a,b] machines of the type ν -SVM Schoelkopf et al. [2000, 2001] with radial basis function kernel were also tested. Because of their relatively high training speed in the case of the small data sets, SVMs were run with ten-fold crossvalidation of the c and γ coefficients. To test the SVMs, the LIBSVM package Chang and Lin [2001] was used. The classification results, averaged over the tested sets, for some other learning machines: C4.5 using classification rules Quinlan [1993], incremental decision tree induction ITI Utgoff [1989], Utgoff and Precup [1997], linear machine decision tree LMDT Utgoff and Brodley [1991], learning vector quantization LVQ Kohonen [1988, 1995], induction of oblique trees OCI Heath et al. [1993], Nevada backpropagation NEVP based on Quickprop Fahlman [1988], k -nearest neighbors with $k = 5$ K5, Q* and radial basis functions RBF Poggio and Girosi [1990], Musavi et al. [1992] were computed using results from tests in Draghici [2001]. A very good comparison of the algorithms can be found in Eklund [2000].

The training limit for the NNs was 10000 iterations. The tested LW networks had considerably more connections to make their single training iteration similarly fast to that of the tested NLW network. In Table 2 results are shown for the classic feedforward networks, the diffused feedforward networks and for ν -SVM with ten-fold crossvalidation of c and γ coefficients. The letter X symbolizes the number of inputs, equal to the number of argument attributes in samples in a given set. The networks LW X-16-16-1 and NLW X-8-8-1 had similar time complexity, but the latter one performed better on average.

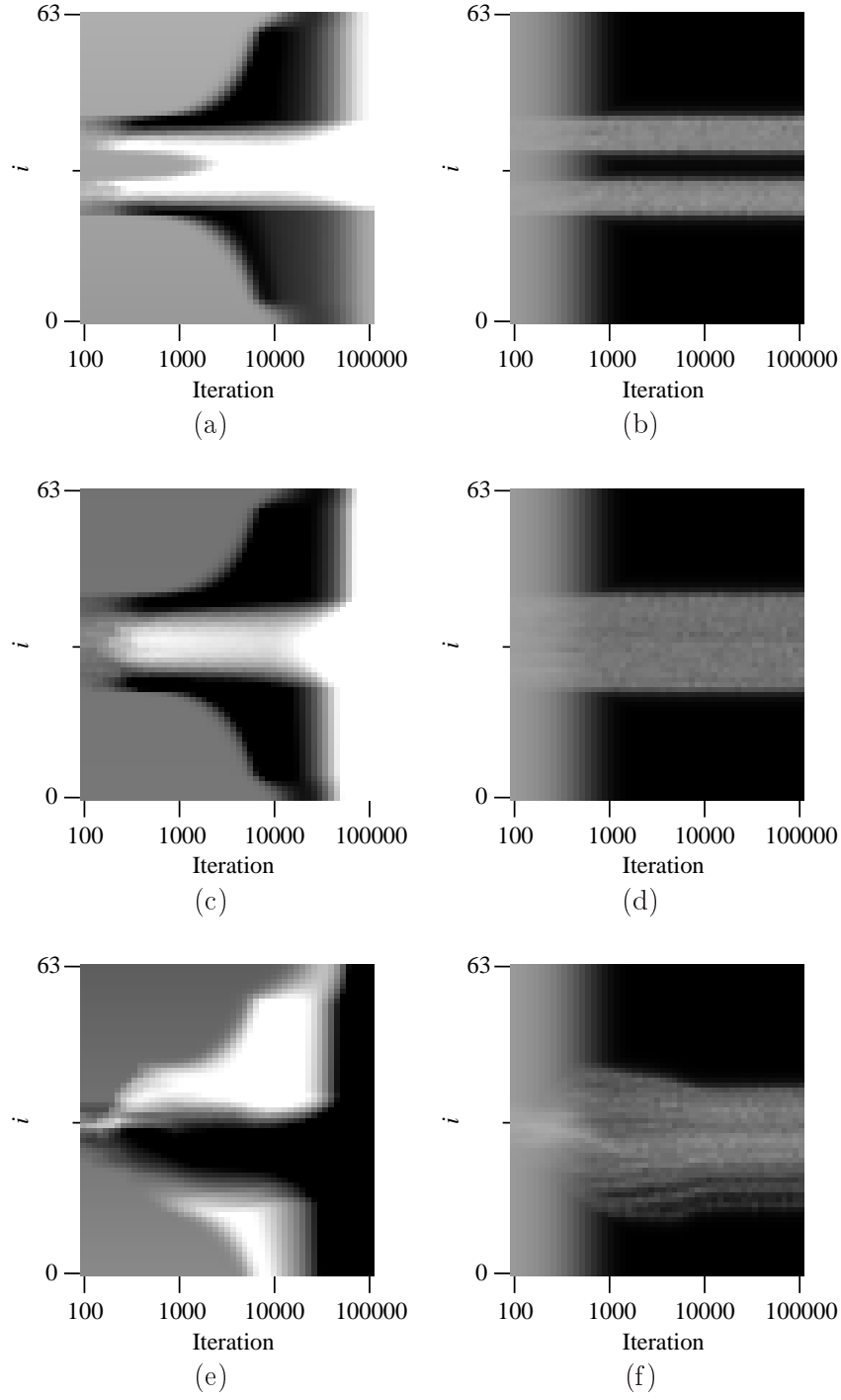


Figure 9: A LUT weights diagram of a 1-1 NN for an image ‘circle’, $R_a^r = 1 \cdot 10^{-4}$, $R_b^r = 1 \cdot 10^{-4}$. Connection $x_0 \rightarrow s(1,0)$ (a) LUT component, (b) visit table, connection $x_1 \rightarrow s(1,0)$ (c) LUT component, (d) visit table, connection $s(1,0) \rightarrow s(2,0)$ (e) LUT component, (f) visit table. The i axes denote weight function LUT (a)(c)(e) or visit table (b)(d)(f) indices, x_j denotes j th input of the NN and $s(l,n)$ denotes a node n in the l th layer.

Table 3 shows the average results for the same data sets, except for RBF neural networks that have the result for the ‘Zoo’ set missing, reported in Draghici [2001], for various other learning machines. In Draghici [2001] detailed results for individual data sets can be found. The comparisons should be interpreted cautiously – the division into the training and test sets may be different, giving various classification results – within the learning machines LW, NLW and SVM tested by the author it was the same, though. It can be seen that SVM had best average results, and NLW was the second in all of the tests. It can also be seen, that the results of NLW networks are relatively similar for very different number of connections and two different r_{res} values.

Data set	LW X-8-1	LW X-16-16-1	LW X-32-32-1	ν -SVM	NLW X-4-1	NLW X-8-8-1	NLW X-16-16-1
					$r_{\text{res}} = 16$	$r_{\text{res}} = 16$	$r_{\text{res}} = 64$
Glass	67.68	66.74	70.93	74.42	76.74	76.05	80.00
Ionosphere	94.71	95.00	95.43	92.86	91.86	93.29	85.89
Wine	98.61	95.28	97.22	97.22	96.39	94.72	95.28
Pima	77.53	77.53	76.56	77.27	78.12	75.91	75.07
Bupa	60.72	59.86	63.63	69.57	60.87	62.03	66.09
Tic tac toe	96.93	96.93	97.04	100.00	96.78	96.20	97.04
Balance	90.64	91.44	90.72	100.00	96.16	96.48	95.68
Iris	96.00	95.33	95.33	96.67	94.67	95.33	94.66
Zoo	86.00	90.50	88.00	85.00	86.50	88.50	87.50
Average	85.42	85.40	86.10	88.11	86.45	86.50	86.36

Table 2: Comparison of classification results of small size data sets for several LW and NLW networks and ν -SVM, in percents.

	C4.5	C4.5r	ITI	LMDT	CN2	LVQ	OC1	NEVP	K5	Q*	RBF	CBD
Average	79.89	82.71	82.25	84.76	83.52	76.97	75.61	82.30	76.68	74.52	75.29	81.94

Table 3: Comparison of average classification results of small size data sets, the same as in Table 2, in percents, for several neural and non-neural learning machines.

4.4 TWO SPIRALS

Because the data sets tested in this section are relatively sparse as for the introduced networks, the resolution of the LUT tables was decreased to $r_{\text{res}} = 16$ and, like it was done in tests in Sec. 4.3, the diffusion was ‘strengthened’ by using relatively large R_b^r values. The NLW networks are compared to the classic ones and to some other learning machines.

Let us first test the generalization of the ‘two spirals’ set, one of the standard benchmarks for learning machines Lang and Witbrock [1988]. This set, after centering around the point (0,0), is shown in Fig. 10(a). Each sample in the set has three attributes (x_0, x_1, y) , the first two being the argument attributes and the last one the value attribute. Even though the spirals may be regarded as relatively well defined because of the density of the samples determining them, this set is known to be a very hard two-class problem to learn by classic feedforward neural networks using the error backpropagation family of learning algorithms. In Lang and Witbrock [1988] it was reported that the

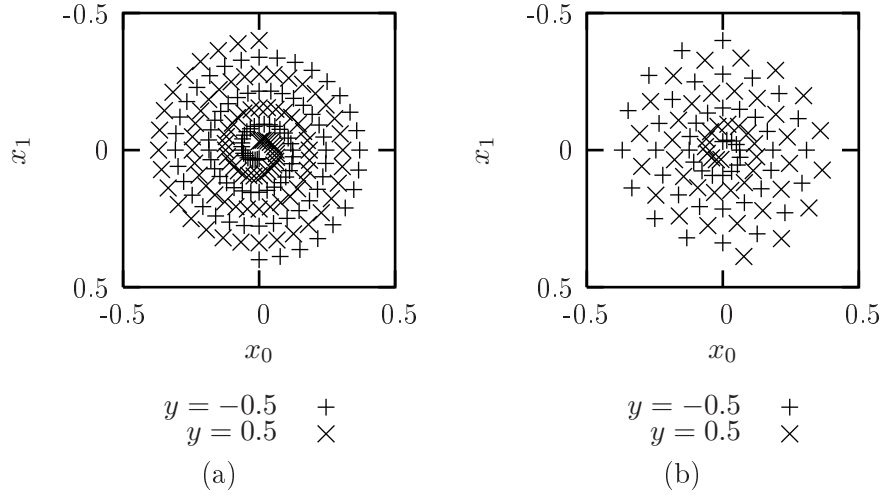


Figure 10: The (a) ‘two spirals’ and (b) ‘two spirals sparse’ data sets.

task could not be solved with the tested classic feedforward networks with connections only between neighboring layers, so to classify each sample in the training set, a special architecture was developed, where each node was connected to all nodes in the subsequent layers and the network was trained using error backpropagation with momentum. Several other trials have been undertaken for improving the learning algorithms to train feedforward neural networks more efficiently. For example in Fahlman and Lebiere [1990] a learning algorithm is developed that grows the trained neural network by adding new trained units to the network. The algorithm was successfully applied to the two spirals problem, but even though the trained neural network learned to classify all samples in the training set, its generalization quality was relatively poor – the decision border was very rough and it even crossed the arms of the spirals in some places. Images of the generalization function of the network can be found also in Fahlman and Lebiere [1990]. The radial basis function neural networks Moody and Darken [1989] even with the advanced techniques like dynamic decay adjustments Berthold and Diamond [1995], show the problem of the lack of a ‘long range’ generalization – as can be seen in the images in Berthold and Diamond [1995], the samples are classified correctly, but the regions far from the samples seem to have little or nothing in common to the positive or negative values of the individual samples. The neural networks presented in Perwass et al. [2003], that have neurons whose decision borders are hyperspheres, have the problem with the ‘long range’ generalization as well, as can be seen in the images in Perwass et al. [2003].

Let the generalization functions of the networks be sampled, and presented as two-dimensional gray scale raster images of the size 64×64 , in the same manner as was done in Sec.4.2. In Fig. 11 generalization functions are shown for NLW networks trained at two different values of the diffusion speed coefficient R_b^r . Fig. 12 shows classification results for these networks and for a ν -SVM Schoelkopf et al. [2000, 2001] The results for the tested LW networks are not shown as they were not able to even classify the training set within $1 \cdot 10^6$ iterations. The SVM performed very good on the set. 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. In Solazzi and Uncini [2000] a neural network is introduced with adaptive

multidimensional spline activation functions, and is shown that the network has excellent results of the generalization results of the two spirals set, similar to these in Fig. 12(a) and (c), if the adaptive spline activation function is two-dimensional. Yet the ‘two spirals’ set has a generalization function that is also two-dimensional. If the adaptive activation functions have a single dimension, like in the networks presented by Uncini, Capparelli and Piazza Uncini et al. [1998], Vecchi, Piazza and Uncini Vecchi et al. [1998], and Guarnieri and Piazza Guarnieri and Piazza [1999], so that the ‘scale up’ of the dimension by the high dimension regressor is non-zero for the ‘two spirals’ set, the images demonstrated in Solazzi and Uncini [2000] shows substantial artifacts, much larger than these in Fig. 12(b).

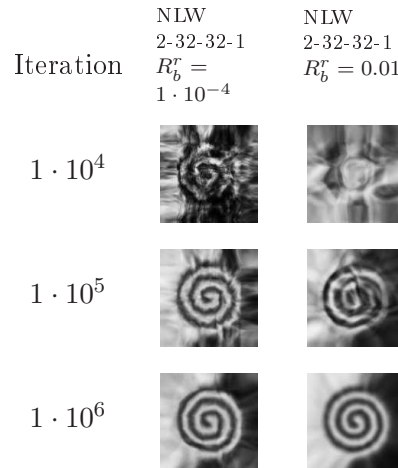


Figure 11: Images of the generalizing function for the ‘two spirals’ set.

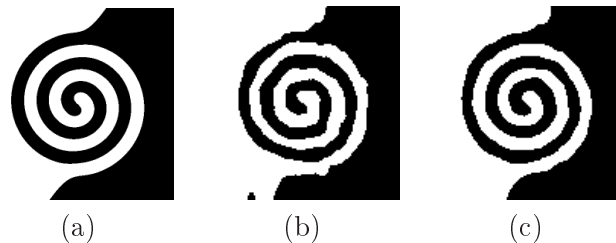


Figure 12: Classification of the ‘two spirals’ set by (a) SVM with radial basis kernel function, $c = 100, \gamma = 150$, and NLW 2-32-32-1 at the $1 \cdot 10^6$ th iteration, at (b) $R_b^r = 1 \cdot 10^{-4}$, (c) $R_b^r = 0.01$.

Let us now discuss another training set, derived from the previous one. Let the samples within each of the spiral arms be counted from the inner beginning of each arm. This set is created by removing each odd sample in one of the spiral arms and each even sample in the other arm. Let the set be called ‘two spirals sparse’. This set is shown in Fig. 10(b). Such a way of removing the samples was used to obtain a special type of patterns in the set. The patterns create two families, ‘along arms’ and ‘radial’, as illustrated in Fig. 13. It can be said that in the inner side of the spirals the ‘along arms’ pattern is stronger than the ‘radial’ one, because of the relationship between appropriate

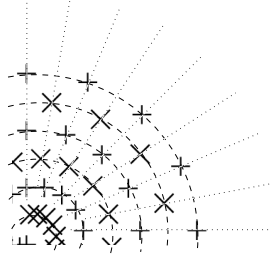


Figure 13: Two families of patterns in the ‘two spirals sparse’ data set, denotes with different types of lines.

distances between samples. Conversely, the ‘radial’ pattern is stronger in the outer regions of the spiral arms. The third type of pattern is created by the outer parts of the spirals. Because the halves are not covered from the outside by any samples, the value attributes of the samples 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 samples.

In Fig. 14 generalization functions at some different iterations for NLW networks trained at two different values of the diffusion speed R_b^r are shown. Fig. 15 shows the classification results for ν -SVM and NLW networks at two different R_b^r values. The

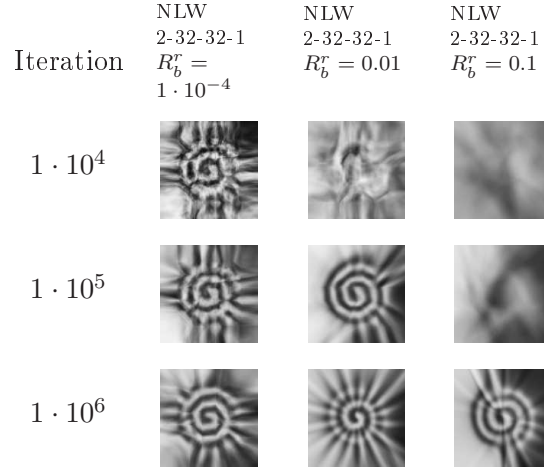


Figure 14: Images of the generalizing function for the ‘two spirals sparse’ set.

tested 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

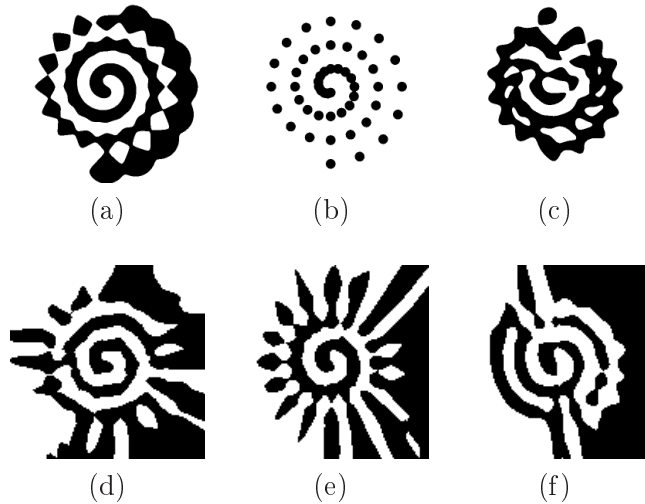


Figure 15: Classification of the ‘two spirals sparse’ set by (a) ν -SVM with radial basis kernel, $c = 1, \gamma = 500$, (b) ν -SVM with radial basis kernel, $c = 1, \gamma = 10000$, (c) one class SVM with radial basis kernel, $c = 100, \gamma = 200$, and NLW 2-32-32-1 at the $1 \cdot 10^6$ th iteration, at (d) $R_b^r = 1 \cdot 10^{-4}$, (e) $R_b^r = 0.01$, (f) $R_b^r = 0.1$.

any improvements – as can be seen in the generalizing functions in Fig.14, the learning process slowed down, and at the $1 \cdot 10^6$ th iteration, the generalizing functions seemed to be relatively less ‘even’. The SVM with the settings like in the previous test with the ‘two spirals’ set, substantially underfitted the ‘two spirals sparse’ set, so it was with tested various different γ values, yet none of the tested SVMs shown results so fine as the NLW networks – SVMs tended to give asymmetric decision borders and to neglect the ‘radial’ pattern, and the one class SVM Schoelkopf et al. [2001] gave particularly spurious results.

4.5 LEARNING A LARGE SIZE SET

In this section, the ‘storage capacity’ of a trained neural network against time is tested, that is the ability to memorize the features in a set, and the speed of the memorizing. Generalization of sets like ‘two spirals’ Lang and Witbrock [1988] shows that even large, and thus slow LW networks might have serious problems with just the memorizing and the speed of the memorizing. The training set tested in this section is relatively complex and, to test the high dimension regressor, the samples have five input attributes each. Standard values of coefficients were used for training the neural networks with this complicated set.

Because the author could not find a standard benchmark of the required complexity and number of samples, custom data set ‘md-2’ was used. Let the generating function of

the set be as follows:

$$\begin{aligned}
y(x_0, x_1, x_2, x_3, x_4) = & \sin(4x_0) \cos(2x_1 + 3x_2) \\
& \left(\left(\frac{\sin(10x_2 + 10x_3) + 1}{2} \right)^{1/2} - 1/2 \right) \\
& \sin(x_3 - 4x_1x_4) \\
& \left(\left(\frac{\sin(10x_0 - 10x_2 + 10x_3) + 1}{2} \right)^{1/2} - 1/2 \right) \\
& \cos(5x_1x_2x_4).
\end{aligned} \tag{28}$$

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

$$x_i = \text{rand}() - 0.5 \quad i = 0, 1, \dots, 4, \tag{29}$$

where $\text{random}()$ is a uniform random number generator, generating values in the range $(0, 1)$. $1.8 \cdot 10^6$ such tuples were used in the training set, and $2 \cdot 10^5$ independently generated tuples were used in the test set. The data set intentionally does not contain noise and is quite densely represented, to test the discussed ‘storage capacity’.

Fig. 16 shows a diagram of mean square error, denoted by MSE, against estimated times for several LW and NLW networks, trained with the set ‘md-2’. It is seen that the

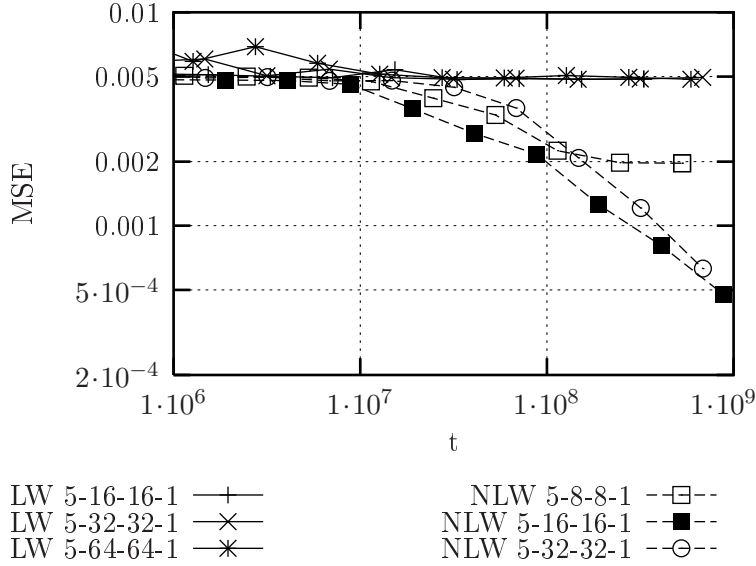


Figure 16: MSE diagram for different neural networks trained with the ‘md-2’ set, against estimated time.

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

5 CONCLUSIONS

The neural networks with diffused weight functions and with the property of selective parameters of the functions showed good performance over a wide range of tested data sets. In particular, they performed very good, in compare to the classic neural networks ant to the tested SVMs, in the case of the subtle patterns in the ‘two spirals sparse’ and ‘camomiles-m’ sets.

References

- M. R. Berthold and J. Diamond. Boosting the performance of rbf networks with dynamic decay adjustment. *Advances in Neural Information Processing Systems*, 7:521–528, 1995.
- C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, New York, 1995.
- C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
- Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001.
- C. de Boor. *A Practical Guide to Splines*. Springer Verlag, 1978.
- S. Draghici. The constraint based decomposition (cbd) training architecture. *Neural Networks*, 14 (4-5):527–550, 2001.
- P. W. Eklund. Comparative study of public-domain supervised machine-learning accuracy on the uci database. In *Proceedings of the Data Mining and Knowledge Discovery*, pages 39–50, 2000.
- S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In *Proceedings of Connectionist Models Summer School*. Morgan Kaufmann, 1988.
- S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, 2, pages 524–532, San Mateo, 1990. Morgan Kaufmann.
- T. N. E. Greville. *Theory and Applications of Spline Functions*. Academic Press, New York, 1969.
- S. Guarnieri and F. Piazza. Multilayer feedforward networks with adaptive spline activation function. *IEEE Transactions on Neural Networks*, 10(2):672–683, 1999.
- S. Guarnieri, F. Piazza, and A. Uncini. Multilayer feedforward networks with adaptive spline activation function. *IEEE Transactions on Neural Networks*, 10(3):672–683, 1999.
- D. Heath, S. Kasif, and S. Salzberg. Induction of oblique decision trees. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1002–1007, 1993.
- J. A. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.
- T. Kohonen. Learning vector quantization. *Neural Networks*, 1, suppl. 1, 1988.
- T. Kohonen. Learning vector quantization. In *The handbook of brain theory and neural networks*, pages 537–540. MIT Press, 1995.
- A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 950–957. Morgan Kaufmann Publishers, Inc., 1992.

- K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59. Morgan Kaufmann Publishers, Inc., 1988.
- W. S. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
- J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.
- M. Musavi, W. Ahmed, K. H. Chan, K. Faris, and D. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5(4):595–603, 1992.
- C. Perwass, V. Banarier, and G. Sommer. Spherical decision surfaces using conformal modelling. In *DAGM-Symposium*, pages 9–16, 2003.
- F. Piazza, A. Uncini, and M. Zenobi. Neural networks with digital lut activation function. In *Proceedings of International Joint Conference on Neural Networks*, volume 2, pages 1401–1404, Nagoya, Japan, 1993.
- T. Poggio and F. Girosi. Networks for approximation and learning. In *Proceedings of the IEEE*, volume 78 (9), pages 1481–1497, 1990.
- J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
- D. E. Rumelhart and J. L. McClelland. Explorations in the microstructure of cognition. *Parallel Distributed Processing*, 1:318–362, 1996.
- B. Schoelkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.
- B. Schoelkopf, A. Smola, R. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.
- M. Solazzi and A. Uncini. Artificial neural networks with adaptive multidimensional spline activation function. In *Proceedings of the IEEE-INNS-ENNS International Conference of Neural Networks*, 2000.
- A. Uncini, F. Capparelli, and F. Piazza. Fast complex adaptive spline neural networks for digital signal processing. In *Proceedings of International Joint Conference on Neural Networks*, pages 903–909, 1998.
- P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.
- P. E. Utgoff and C. E. Brodley. Linear machine decision trees. Technical Report UM-CS-1991-010, University of Massachussets, Computer Science, 1991.
- P. E. Utgoff and D. Precup. Constructive function approximation. Technical Report 97-04, Department of Computer Science, University of Massachussets, 1997.
- V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995a.
- V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1995b.
- L. Vecchi, F. Piazza, and A. Uncini. Learning and approximation capabilities of adaptive spline activation neural networks. *Neural Networks*, 11(2):259–270, 1998.