Appl. Math. and Comp. Sci., 1994, vol.4, No.3, 281–307

MULTILAYER PERCEPTRON NETWORKS: SELECTED ASPECTS OF TRAINING OPTIMIZATION†

JACEK M. ŻURADA*, ALEKSANDER MALINOWSKI*

Training of Multilayer Perceptron Neural Networks using the popular error back propagation method can be modified and its performance improved. The modified original generalized delta learning rule has been found to considerably enhance the learning process. In addition, input layer size can be reducible through evaluation of the network sensitivity over the training/test data set. Minimum set size estimation based on the sampling theorem can also be performed to determine the optimum number of training patterns.

1. Introduction

Multilayer Perceptron Neural Networks (MLPNN) account for the majority of today's applications of neural networks. They are used to express functional relationships between the sets of experimental data describing process identification and modelling of function, approximation, classification, prediction, one-step ahead control, and other tasks. MLPNN's ability to model experimental relationships and to embed knowledge from data into networks is due to the stochastic approximation based on learning within a rather flexible architecture. Although a layered architecture of an MLPNN with a single hidden layer fits into modelling of numerous tasks, a number of questions as to how to optimize the training and the network itself remain unanswered.

One important issue concerns the optimality of the generalized delta learning rule (EBPT). Although the EBPT algorithm has been widely used and hundreds of technical reports illustrate its successful applications, the lambda learning rule often offers a considerable improvement of learning. The paper outlines the generalized lambda learning algorithm for layered networks. It also focuses on visualization of learning and draws comparisons between the two learning approaches.

Minimization of redundancy in the training data is another important issue. When certain inputs bear none, or little, statistical or deterministic relationships to outputs, input vectors can be compressed. This makes it possible to reduce the dimensionality of the input vector, x, through pruning of the input data set, so that a smaller network can be utilized as a model of relationship between the data. Initial findings on this subject have been published in (Cloete and Ludik, 1993; Fu and Chen, 1993; Żurada and Malinowski, 1993). This paper introduces a more formal approach to reduction of input size of the network.

† This work was supported in part by the ONR Grant N00014-93-1-0855

* Department of Electrical Engineering, University of Louisville, Louisville, Kentucky 40292, USA

A new approach to the problem of n-dimensional function approximation using MLPNN is also discussed. The generalized Nyquist theorem is used to look for the optimum number of learning patterns in n-dimensional input space. Choosing the smallest but still sufficient set of training vectors results in a reduced number of hidden neurons and learning time for the network. Analytical formulae and an algorithm for training set size reduction are developed and illustrated by two-dimensional data examples.

2. Lambda Learning Rule for MLPNN

Assume that a non-augmented input vector $\boldsymbol{x}$ has $n-1$ components. Lambda learning rule involves expansion of the learning space to $n+1$ dimensions. In this rule, in addition to weight learning, the steepness of the activation function undergoes adjustment in the negative gradient direction. The first observation of this useful property has been made in (Tawel, 1989) and, independently, in (Movellan, 1987). A number of simulations have confirmed that the method not only accelerates learning, but also improves generalizations and introduces a better participation of nodes in representing the input (Kruschke and Movellan, 1991).

Using the customary expression for error between the desired output value d and the actual one o, i.e.

$$E(\omega, \lambda) = \frac{1}{2}|d - o(\omega, \lambda)|^2 \tag{1}$$

we obtain the following weight and λ adjustments for a single neuron learning

$$\Delta\omega_i = \eta_1 \frac{\partial E}{\partial \omega_i} = -\eta_1 (d-o) f'(\lambda net)\lambda x_i \tag{2}$$

$$\Delta\lambda = \eta_2 \frac{\partial E}{\partial \lambda} = -\eta_2 (d-o) f'(\lambda net) net \tag{3}$$

where $f'(net) = o(1-o)$ and the activation value and function are, respectively,

$$net = \boldsymbol{w}^T \boldsymbol{y}$$

$$f(\lambda, net) = (1 + e^{-\lambda net})^{-1} \tag{4}$$

and η_1, η_2 are positive learning constants usually selected of arbitrary small values. Noticeably, expression (2) coincides exactly with the delta learning rule. Additional weight adjustment as in (3) is the essence of the lambda rule. It expresses such block changes of the value net that although the individual weights remain invariant, the overall impact of learning typically increases, since the adaptive gain factor in the exponent of (4) is suitably adjusted. This feature may be of particular importance when all weights are considerably too small or too large, and/or the neuron learning progresses slowly. Intuitively, (3) corresponds to a separate and independent weight scaling step, when all the multiplicative weights are adjusted up or down in a block-wise fashion in a direction which minimizes the current training error.

The lambda learning rule can now be easily generalized for a single or double layer learning. Assume that the output layer of a two-layer network undergoes the training. Here, steepness coefficients λ_i, $i = 1, 2, \ldots, K$, undergo adjustments in addition to the individual weights. With reference to Fig. 1, the adjusted learning variables can be obtained from (2), (3) as below (primes denote updated values)

$$\omega'_{kj} = \omega_{kj} + \eta_1(d_k - o_k)f'(net_k)\lambda_k x_j \tag{5}$$

$$\lambda'_k = \lambda_k + \eta_2(d_k - o_k)f'(net_k)net_k \tag{6}$$

where $j = 1, 2, \ldots, J$, and k is the neuron number, weights of which undergo adjustments. Understandably, k must be incremented from 1 through K in order for the entire layer to update both its weights and λ values for a single input pattern.

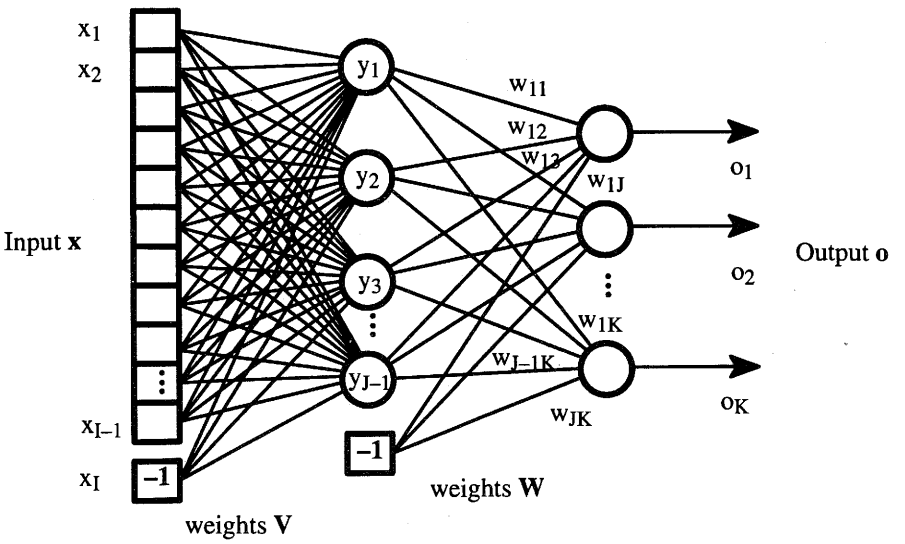


Fig. 1. MLPNN for illustration of lambda and generalized lambda learning.

The generalization below refers to the hidden layer learning, specifically, to learning of weights v_{ji} using the notation of Fig. 1. Obviously, the learning of the j-th neuron is performed such that v_{ji} and λ_j, $i = 1, 2, \ldots, I$, and $j = 1, 2, \ldots, J$, are adapted.

Using the following overall output error definition

$$E = \frac{1}{2}\sum_{k=1}^{K}(d_k - o_k)^2 \tag{7}$$

we obtain for weights and steepness coefficients the adjustments

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial v_{ji}} \tag{8}$$

$$\frac{\partial E}{\partial \lambda_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \lambda_j} \tag{9}$$

Further rearrangements using standard notation (Żurada, 1992) and based on derivations detailed in (Żurada, 1993) lead to the following weight adjustments in the generalized lambda learning rule

$$\Delta \nu_{ji} = \eta_1 f'(net_j) z_i \lambda_j \sum_{k=1}^{K} \delta_{ok} \omega_{kj} \tag{10}$$

$$\Delta \lambda_j = \eta_2 f'(net_j) net_j \sum_{k=1}^{K} \delta_{ok} net_k \tag{11}$$

where $\delta_{ok} = (d_k - o_k) f'(net_k)$.

3. MLPNN Training Algorithm with Lambda Learning

The complete learning algorithm for the network of Fig. 1 is given below:

BEGIN: Given are training pairs of vectors of inputs and desired outputs $\{\boldsymbol{z}_1, \boldsymbol{d}_1, \boldsymbol{z}_2, \boldsymbol{d}_2, \ldots, \boldsymbol{z}_p, \boldsymbol{d}_p\}$ where $\boldsymbol{z}_i$ is $(I \times 1)$, $\boldsymbol{d}_i$ is $(K \times 1)$ and $i = 1, 2, \ldots, P$. Note that the I-th component of each $\boldsymbol{z}_i$ is of value -1 since input vectors are augmented. Size $J - 1$ of the hidden layer having outputs $\boldsymbol{y}$ is selected. Note that the J-th component of $\boldsymbol{y}$ is of value -1, since hidden layer outputs are also augmented; $\boldsymbol{y}$ is $(J \times 1)$ and $\boldsymbol{o}$ is $(K \times 1)$.

STEP 1: η_1, $\eta_2 > 0$, acceptable training error E_{max} chosen. Weights $\boldsymbol{W}$ and $\boldsymbol{V}$ are initialized at small random values; $\boldsymbol{W}$ is $(K \times J)$, V is $(J \times I)$, $q := 1$, $p := 1$, $E := 0$

STEP 2: Training step starts here. Input is presented and the layers' outputs are computed as in (1):

$$z := z_p, \qquad d := d_p, \qquad y_j := f(\boldsymbol{v}_j^T \boldsymbol{z}) \quad \text{for } j = 1, 2, \ldots, J$$

where $\boldsymbol{v}_j$, a column vector, is the j-th row of $\boldsymbol{V}$, and

$$o_k := f(\boldsymbol{w}_k^T \boldsymbol{y}) \qquad \text{for } k = 1, 2, \ldots, K$$

where $\boldsymbol{w}_k$, a column vector, is the k-th row of $\boldsymbol{W}$.

STEP 3: Error value is computed: $E := 0.5(d_k - o_k)^2 + E$, for $k = 1, 2, \ldots, K$.

STEP 4: Error signal vectors δ_o and δ_y of both layers are computed. Vector δ_o is $(K \times 1)$, δ_y is $(J \times 1)$.

The error signal terms of the output and hidden layers in this step are, respectively

$$\delta_{ok} := (d_k - o_k)(1 - o_k) o_k \qquad \text{for } k = 1, 2, \ldots, K$$

$$\delta_{yj} := y_j(1 - y_j) \sum \delta_o w_{kj} \quad \text{for} \quad j = 1, 2, \ldots, J, \quad \text{over} \quad k = 1, 2, \ldots, K$$

STEP 5: Output layer weights and gains are adjusted:

$$w_{kj} := w_{kj} + \eta_1 \delta_{ok} \lambda_j$$

$$\lambda_k := \lambda_k + \eta_2 \delta_{ok} net_k \quad \text{for} \quad k = 1, 2, \ldots, K \quad \text{and} \quad j = 1, 2, \ldots, J$$

STEP 6: Hidden layer weights and gains are adjusted:

$$v_{ji} := v_{ji} + \eta_1 \delta_{yj} \lambda_j z_i$$

$$\lambda_j := \lambda_j + \eta_2 \delta_{yj} net_j \quad \text{for} \quad j = 1, 2, \ldots, J \quad \text{and} \quad i = 1, 2, \ldots, I$$

STEP 7: If $p < P$, then $p := p + 1$, $q := q + 1$, and go to Step 2; otherwise go to Step 8.

STEP 8: The training cycle is computed. For $E < E_{max}$ terminate the training session. Output weights $\boldsymbol{W}$, $\boldsymbol{V}$, and the cycle counter q, as well as the error E. If $E > E_{max}$, then $E := 0$, $p := 1$, and initiate the new training cycle by going to Step 2.

END.

4. Experimental Results

Simulation of learning efficiency using the lambda learning rule has been tested on a number of cases, including a bit-map classifier. Although the lambda learning rule-based algorithm is numerically about as complex as the classical EBPT, it can offer an improvement in the efficiency of training. Since the improvement comes at about no additional cost, the algorithm seems to be a promising one and likely to become important. The same learning constants and initial weights have been used for comparison of both algorithms (simulations are made for bipolar neurons' characteristics).

Learning using the EBPT and the lambda learning rule have been simulated and compared for the XOR problem. Figure 2 shows two typical learning profiles for the EBPT and lambda learning rule, and it indicates faster learning for the method employing adaptive gain. Two hidden-layer neurons have been used in this experiment. Figure 3 illustrates gain variations during training and corresponds to the bottom error curve of Fig. 2. It can be seen that the output neuron's gain changes sign during learning and its gain variations are substantially correlated with an associated error curve of Fig. 2.

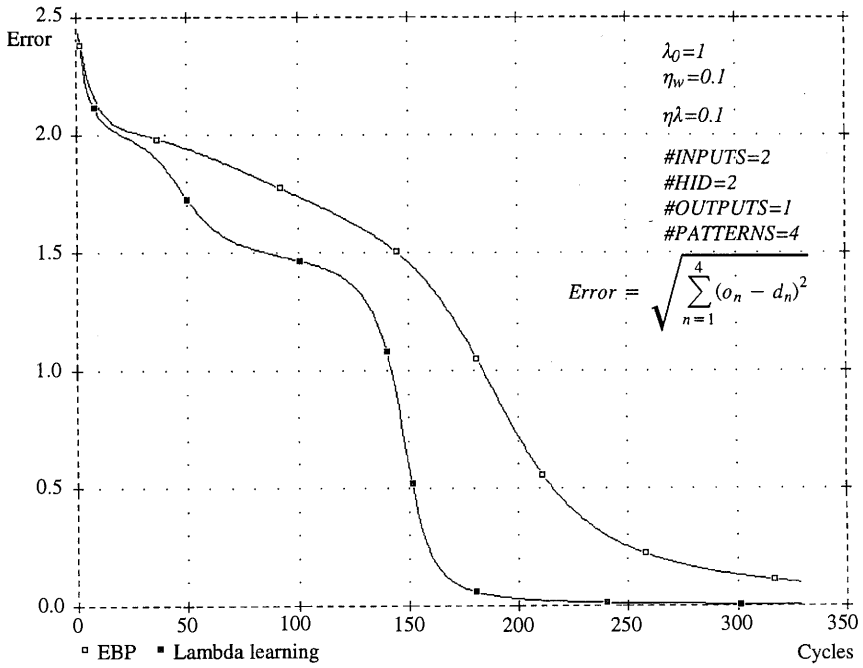


Fig. 2. Learning profile for the XOR problem.

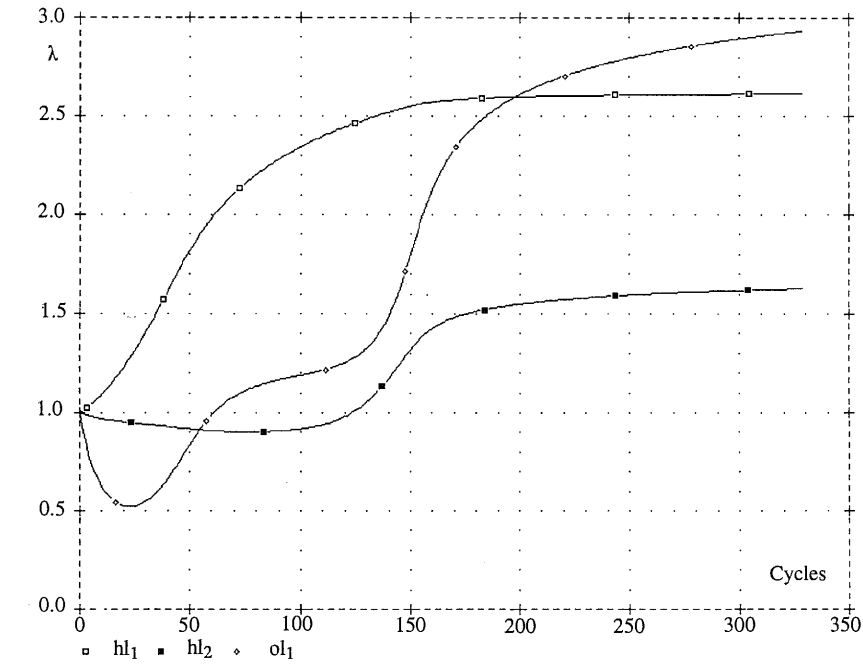


Fig. 3. Neurons' lambda variations during λ-learning for the XOR problem.

Efficiency of both algorithms has also been compared for training of a 36-class classifier of digits 0–9 and 26 letter characters with 8×10 binary input pixel field. The desired output vector has been the 7-bit ASCII code of each character. The architecture with 30 hidden nodes have been used. Figure 4 depicts two typical learning profiles produced for this application and indicates that the lambda learning method yields several times faster learning for the same final level of classifier's performance. Noticeably, Fig. 4 illustrates one case in which the EBPT method has not been successful and the lambda learning algorithm was.

In addition to the typical simulation results discussed in this section, some additional observations are in place. The lambda learning algorithm seems to be rather sensitive to initial weights and values. For small networks, such as XOR, $\lambda = 1$ and $\eta = 0.1$ yield seemingly best results. Initial weights uniformly spread within $\pm 1/\sqrt{(fan - in)}$ yield satisfactory results for initial value of $\lambda = 1$. It has also been found experimentally that both methods with weights initialized improperly result in persistent saturation of weights. In such a case no further learning occurs despite large output error value. Many simulations also indicated that lambda learning has led to quick and excellent solutions in cases when the standard EBPT has failed to produce acceptable results in reasonable time.

5. Sensitivity-based Approach for Input Vector Reduction

The minimization of redundancy which may be present in the training data is an important issue and rather rarely addressed in the technical literature. MLPNN are often used to model complex functional relationships between sets of experimental data. As such, they perform as function approximators which learn on a set of training patterns/examples (Hartman *et al.*, 1990; Shekhar and Amin, 1992).

Let us define the sensitivity of a trained MLPNN output o_k, with respect to its input x_i as

$$S_{x_i}^{o_k} \doteq \frac{\partial o_k}{\partial x_i} \tag{12}$$

which can be written succinctly as

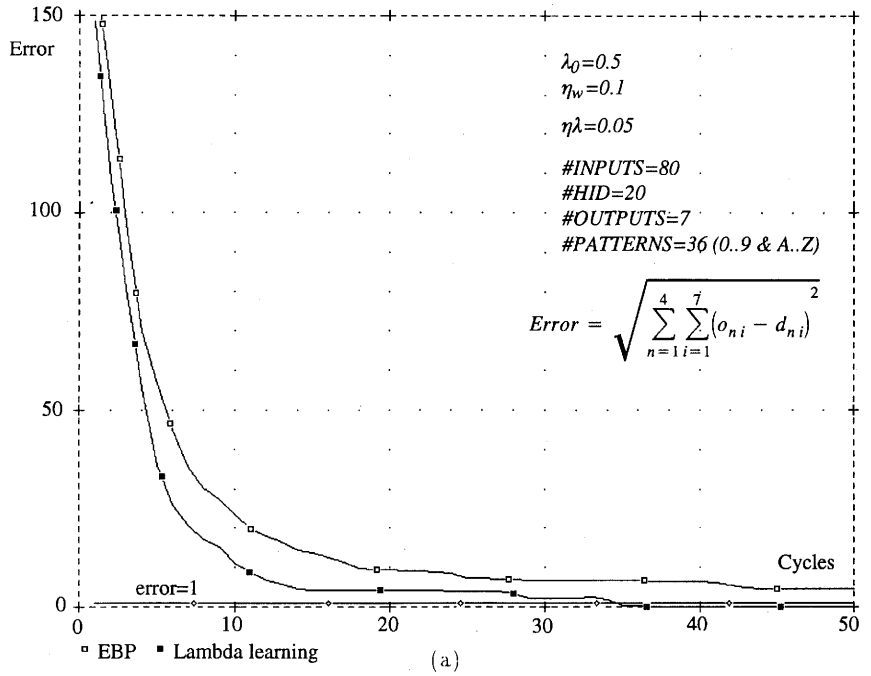
$$S_{ki} \doteq S_{x_i}^{o_k} \tag{13}$$

By using standard notation of the EBPT, the derivative of (12) can be readily expressed in terms of network weights as follows

$$\frac{\partial o_k}{\partial x_i} = o_k' \sum_{j=1}^{J-1} w_{kj} \frac{\partial y_j}{\partial x_i} \tag{14}$$

where y_j denotes the output of the j-th neuron of the hidden layer, and o_k' is the value of derivative of the activation function $o = f(net)$ at the k-th output neuron. This further yields

$$\frac{\partial o_k}{\partial x_i} = o_k' \sum_{j=1}^{J-1} w_{kj} y_j' \nu_{ji} \tag{15}$$



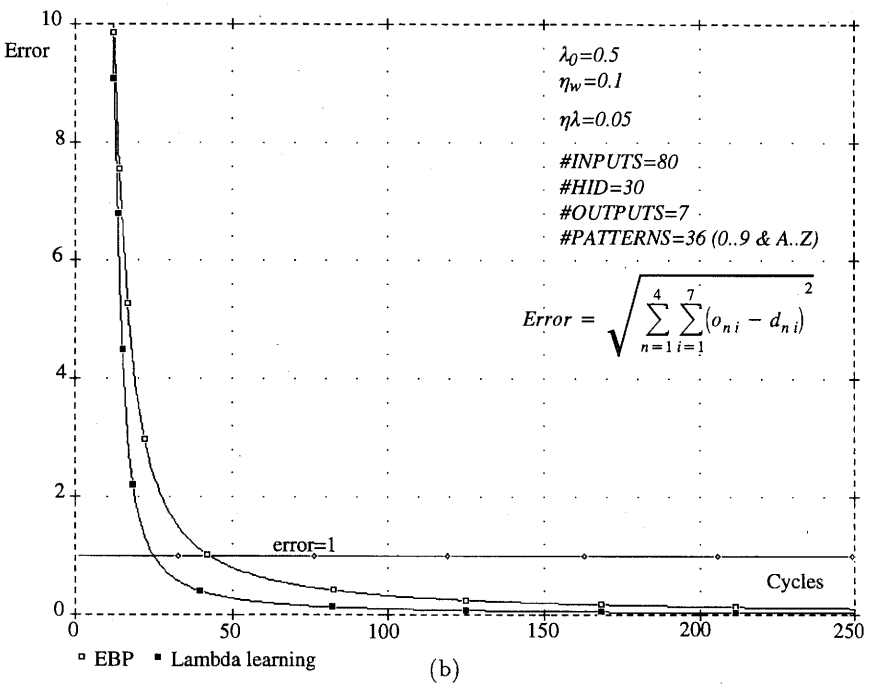


Fig. 4. Learning profiles for character classifiers (a) 20 hidden neurons (b) 30 hidden neurons.

where y'_j is the value of derivative of the activation function $y = f(net)$ of the j-th hidden neuron ($y'_J = 0$ since the J-th neuron is a dummy one, i.e. it serves as a bias input to the output layer). The sensitivity matrix $\boldsymbol{S}(K \times I)$ consisting of entries as in (15) or (13) can now be expressed using array notation as

$$\boldsymbol{S} = \boldsymbol{O}' \times \boldsymbol{W} \times \boldsymbol{Y}' \times \boldsymbol{V} \tag{16}$$

$\boldsymbol{W}(K \times J)$ and $\boldsymbol{V}(J \times I)$ are output and hidden layer weight matrices, respectively, and $\boldsymbol{O}'(K \times K)$ and $\boldsymbol{Y}'(J \times J)$ are diagonal matrices defined as follows

$$\begin{aligned} \boldsymbol{O}' &\doteq \operatorname{diag}(o'_1, o'_2, \ldots, o'_K) \\ \boldsymbol{Y}' &\doteq \operatorname{diag}(y'_1, y'_2, \ldots, y'_J) \end{aligned} \tag{17}$$

Matrix $\boldsymbol{S}$ contains entries S_{ki} which are ratios of absolute increments of output k due to the input i as defined in (13). This matrix depends only upon the network weights as well as slopes of the activation functions of all neurons. Each training vector $\boldsymbol{x}^{(n)} \in \mathcal{X}$, where $\mathcal{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}\}$ denotes the training set, produces different sensitivity matrix $\boldsymbol{S}^{(n)}$ even for a fixed network. This is due to the fact that although weights of a trained network remain constant, the activation values of neurons change across the set of training vectors $\boldsymbol{x}^{(n)}$, $n = 1, 2, \ldots, N$. This, in turn, produces different diagonal matrices of derivatives $\boldsymbol{O}'$ and $\boldsymbol{Y}'$, which strongly depend upon the neurons' operating points determined by their activation values.

In order to possibly reduce the dimensionality of input vectors, the sensitivity matrix as in (16) needs to be evaluated over the entire training set $\mathcal{X}$. Let us define the sensitivity matrix for the pattern $\boldsymbol{x}_n$ as $\boldsymbol{S}^{(n)}$. There are several ways to define the overall sensitivity matrix, each relating to the different objective functions which need to be minimized.

The *mean square average sensitivities*, $S_{ki,avg}$, over the set $\mathcal{X}$ can be computed as follows

$$S_{ki,avg} \doteq \sqrt{\frac{1}{N} \sum_{n=1}^{N} \left(S_{ki}^{(n)}\right)^2} \tag{18}$$

Matrix $\boldsymbol{S}_{avg}(K \times I)$ is defined as $\boldsymbol{S}_{avg} = [S_{ki,avg}]$. This method of sensitivity averaging is coherent with the goal of network training which minimizes the mean square error over all outputs and all patterns in the set.

The *absolute value average sensitivities*, $S_{ki,abs}$, over the set $\mathcal{X}$ can be computed as follows

$$S_{ki,abs} \doteq \sqrt{\frac{1}{N} \sum_{n=1}^{N} \left|S_{ki}^{(n)}\right|} \tag{19}$$

Matrix $\boldsymbol{S}_{abs}(K \times I)$ is defined as $\boldsymbol{S}_{abs} = [S_{ki,abs}]$. Note that summing sensitivities across the training set requires taking their absolute values due to the possibility of

cancelations of negative and positive values. This method of averaging may be better than (18) if sensitivities $S_{ki}^{(n)}$, $n = 1, \ldots, N$, are of disparate values.

The *maximum sensitivities*, $S_{ki,max}$, over the set $\mathcal{X}$ can be computed as

$$S_{ki,max} \doteq \max_{n=1,\ldots,N} \left\{ S_{ki}^{(n)} \right\} \tag{20}$$

Matrix $\boldsymbol{S}_{max}(K \times I)$ is defined as $\boldsymbol{S}_{max} = [S_{ki,max}]$. This sensitivity definition allows us to prevent deleting inputs which are relevant only in small percentage of inputs.

Any of the sensitivity measure matrices proposed in (18)–(20) can provide useful information as to the relative significance of each of the inputs in $\mathcal{X}$ to each of the outputs. For the sake of simplicity, however, mainly the matrix defined in (18) will be applied in further discussion. The cumulative statistical information resulting from (18) will be used along with criteria for reducing the number of inputs to the smallest number sufficient for accurate learning.

6. Algorithm for Pruning Inputs

Inspection of the average sensitivity matrix $\boldsymbol{S}_{avg}$ renders it possible to determine which inputs affect outputs least. A small value of $S_{ki,avg}$ in comparison to others means that for the particular k-th output of the network, the i-th input does not significantly contribute to output k, and may therefore be possibly disregarded. This property allows the formulation of the following pruning rule: *The sensitivity matrices for a trained neural network can be evaluated for both training and testing data sets; the values of average sensitivity matrix entries can be used for determining the least significant inputs and for reducing the size of network accordingly by pruning redundant inputs.*

When one or more of the inputs have relatively small sensitivity in comparison to others, the dimension of neural network can be reduced by suppressing them, and smaller-size neural network can be successfully used in most cases. The criterion used below for determining which inputs can be pruned is based on the so-called the largest gap method.

In order to normalize the data relevant for comparison of the significance of inputs, the sensitivity matrices defined in (18)–(20) have to be additionally preprocessed. The formulae needed for scaling are given in (21) and map each input into range [0, 1] as well as each output into range [−1, 1]:

$$\begin{aligned} \widehat{x}_i^{(m)} &\doteq \frac{x_i^{(m)} - \min\limits_{n=1,\ldots,N} \left\{ x_i^{(n)} \right\}}{\left(\max\limits_{n=1,\ldots,N} \left\{ x_i^{(n)} \right\} - \min\limits_{n=1,\ldots,N} \left\{ x_i^{(n)} \right\} \right)} \\ \widehat{o}_k^{(m)} &\doteq \frac{o_k^{(m)} - \left(\max\limits_{n=1,\ldots,N} \left\{ o_k^{(n)} \right\} + \min\limits_{n=1,\ldots,N} \left\{ o_k^{(n)} \right\} \right)/2}{\left(\max\limits_{n=1,\ldots,N} \left\{ o_k^{(n)} \right\} - \min\limits_{n=1,\ldots,N} \left\{ o_k^{(n)} \right\} \right)} \end{aligned} \tag{21}$$

If input and output data scaling (21) was performed before network training, no additional operations on S_{ki} are required and we have

$$\widehat{S}_{ki,avg} \doteq S_{ki} \tag{22}$$

Note that the scaling can be performed either on entries of $\boldsymbol{S}$ or $\boldsymbol{S}_{avg}$. In the case when network original inputs and outputs are not scaled to the same level, additional scaling is necessary to allow for accurate comparison among inputs:

$$\widehat{S}_{ki,avg} \doteq S_{ki} \frac{\left(\max\limits_{n=1,\ldots,N} \{x_i^{(n)}\} - \min\limits_{n=1,\ldots,N} \{x_i^{(n)}\} \right)}{\left(\max\limits_{n=1,\ldots,N} \{o_k^{(n)}\} - \min\limits_{n=1,\ldots,N} \{o_k^{(n)}\} \right)} \tag{23}$$

The significance of the i-th input Φ_i across the entire set $\mathcal{X}$ is defined as follows:

$$\Phi_{i,avg} \doteq \max_{k=1,\ldots,K} \{\widehat{S}_{ki,avg}\} \tag{24}$$

In order to distinguish inputs, entries of Φ have to be sorted in descending order so that

$$\Phi_{i_m+1} \geq \Phi_{i_m}, \quad m = 1, \ldots, I-1 \tag{25}$$

where i_m is a sequence of sorted input numbers. Let us define the measure of gap as

$$g_{i_m} \doteq \frac{\Phi_{i_m}}{\Phi_{i_m+1}} \tag{26}$$

and then find the largest gap using the following formula

$$g_{max} \doteq \max_{i_m} \{g_{i_m}\} \quad \text{and} \quad m_{CUT} \doteq m \quad \text{such that} \quad g_{i_m} = g_{max} \tag{27}$$

If condition (28) below is valid, then the gap found between m_{CUT} and m_{CUT+1} is large enough:

$$C g_{max} > \max_{i_m \neq i_{m_{CUT}}} \{g_{i_m}\} \tag{28}$$

Constant C from (28) is chosen arbitraily within a reasonable range (e.g. $C = 0.5$; the smaller C, the stronger is the condition for existence of an acceptable gap). All inputs with indexes $\{i_{m+1} \ldots i_{I-1}\}$ can be pruned with the smallest loss of information to the MLPNN.

The gap method can be also applied for comparison among sensitivities of inputs to each output separately. For this purpose, a set containing candidates for pruning can be created for every output. Final pruning is then performed by removing these inputs which can be found in every set determined previously for each output independently.

$\boldsymbol{S}_{avg}$ can be meaningfully evaluated only for well trained neural networks. Despite this condition, it can still save computational effort when initial learning can be performed on smaller, but still representative subset of data. $\boldsymbol{S}_{avg}$ can be evaluated

based either on the data set used for initial training or on the complete data set. Subsequently, newly developed neural network with appropriate inputs can be retrained using the full set of training patterns of reduced dimension.

7. Experimental Results

A series of numerical simulations was performed in order to verify the proposed definitions and pruning criteria. In the first experiment a training set for a neural network was generated using four inputs $x_1, \ldots, x_4$ and two outputs o_1 and o_2. Values of outputs were correlated with x_1 and x_2 for o_1, and with x_2 and x_3 for o_2. Input vectors $\boldsymbol{x}$ (4×1) were produced using a random number generator. The expected values of vector $\boldsymbol{d}$ (2×1) for the output vector $\boldsymbol{o}$ (2×1) were evaluated for each $\boldsymbol{x}$ using a known relationship $\boldsymbol{d} = \boldsymbol{F}(\boldsymbol{x})$ where $\boldsymbol{d}$ is the desired (target) output vector for supervised training. The training set $\mathcal{X}$ consisted of $N = 81$ patterns. A neural network with 4 inputs, 2 outputs and 6 hidden neurons ($I = 5$, $J = 7$, $K = 2$) has been trained for the mean square error defined as follows

$$MSE \doteq \sqrt{\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \left(d_k^{(n)} - o_k^{(n)} \right)^2} \tag{29}$$

equal to 0.001 per input vector. Matrices of sensitivities were subsequently evaluated and $\boldsymbol{S}_{avg}$ produced at the end of training over the entire input data set $\mathcal{X}$.

The changes of sensitivity entries during learning are presented in Fig. 5.

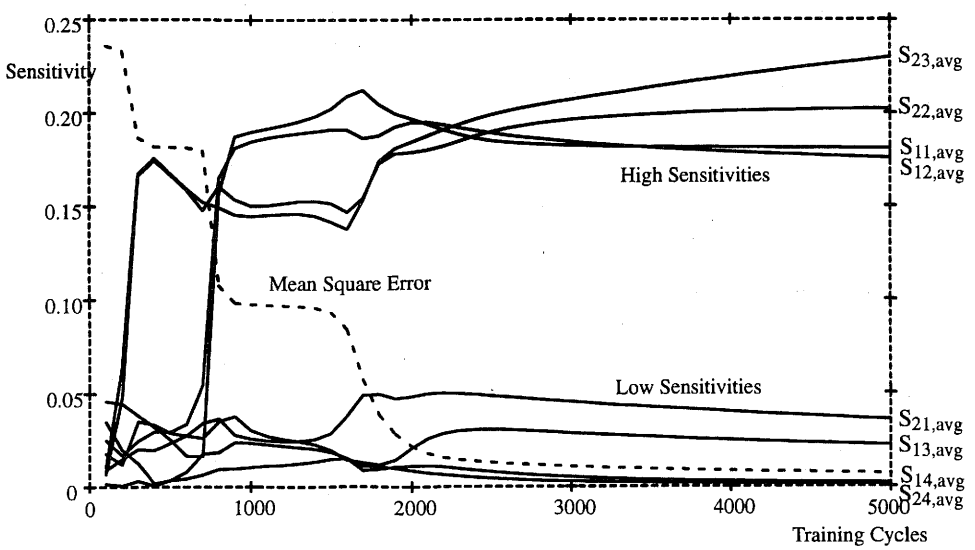


Fig. 5. Sensitivity profile for the full training data set.

It can be seen that an untrained neural network in the example has per average smaller sensitivities than after the training. During the training some of the average

sensitivities $S_{ki,avg}$ increase, while the others converge towards low values. Final values of sensitivities of the first output offer hints for deleting x_3 and x_4, and these for the second output indicate that x_1 and x_4 could be deleted. The only input which occurs in both sets of candidates for deletion is x_4. Therefore, the fourth input to the network can be skipped and its dimension reduced to 3 ($I = 4$).

The second experiment was performed using a larger network and fuzzy data. MLPNN had 20 inputs ($I = 21$), 10 hidden neurons ($J = 26$) and 4 outputs ($K = 4$). There were $N = 500$ patterns in the training set and several additional data sets of the same size for network performance evaluation. The network was successfully trained to the MSE error of 0.15. However, due to the fuzziness of the training data, MSE error for additional sets remained at the level of 0.20.

All outputs were strongly correlated with inputs x_1, x_2, x_3, x_4, x_6, x_8, and x_9. Input x_6 during data generation was multiplied by random numbers, while the influence of x_2 and x_4 on outputs was scaled down to remain small in comparison with other inputs (less than 0.05).

Input significance coefficients calculated using formulae (18)–(20) are shown in Fig. 6.

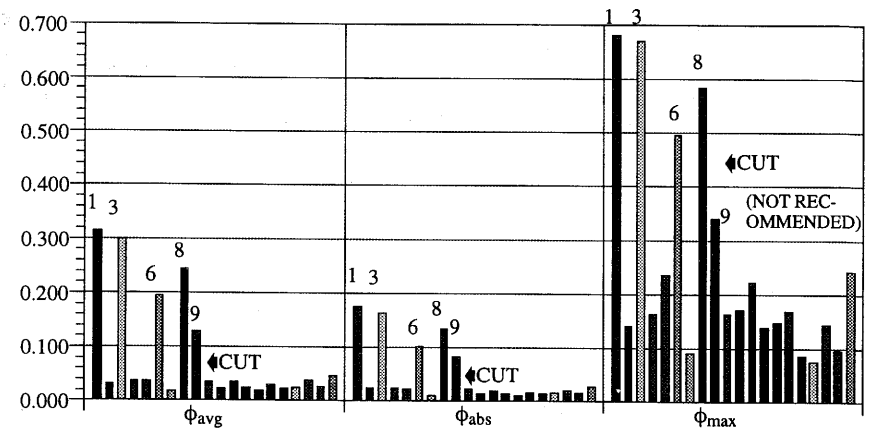


Fig. 6. Input significance coefficients ϕ for different sensitivities (18)–(20) and pruning criterion (28).

Inputs x_2 and x_4 are identified after sorting as less important than other, not correlated inputs. This is due to their low correlation with outputs. They therefore can be ignored along with other inputs which are not correlated for a given MSE error value. The sequence of significance is the same for all proposed methods, however, the sizes of gaps are different in each case. Value $C = 0.5$ prevents pruning using ϕ_{max} definition. Note that the maximum method does not give the clear clue how to choose the significance level for purging due to fuzziness of the training data.

The result of initial training is shown in Fig. 7. It can be determined from this figure which inputs should remain after pruning. The network performance after

pruning is shown in Fig. 8. No further input dimensionality reduction is possible since no large gap in coefficients ϕ can be found. The speed of training has increased mostly because of reduction of the MLPNN size (input dimension reduced by 4). The necessary number of the training cycles has also decreased, but not so dramatically as in the first experiment.

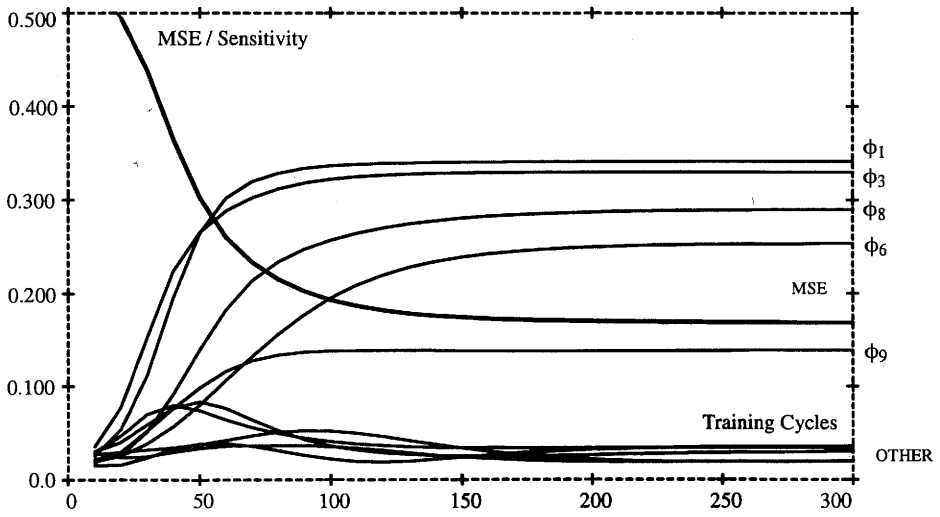


Fig. 7. Input significance ϕ_{avg} for the full training set.

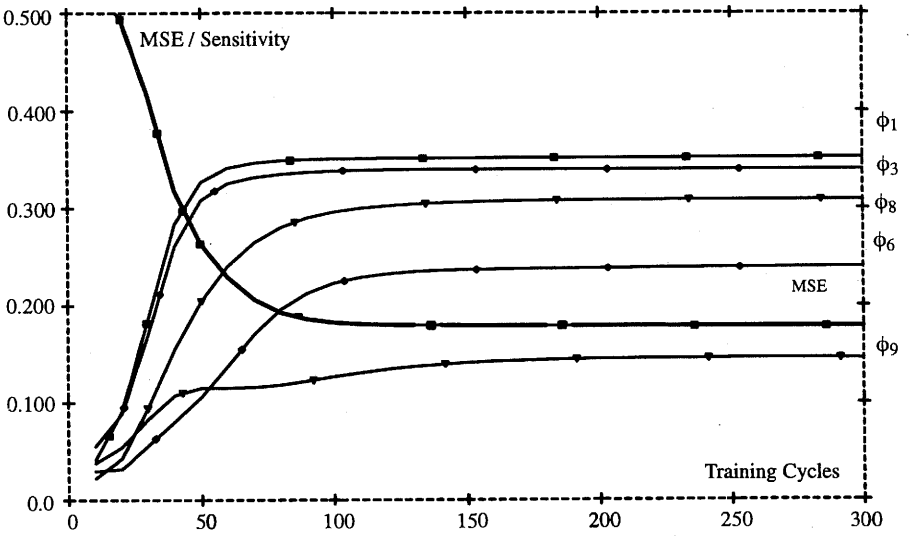


Fig. 8. Input significance ϕ_{avg} for the pruned training set.

Using the sensitivity-based approach for input layer pruning seems particularly useful when network training requires large amount of redundant, oversized data. In the first phase, network can be pre-trained until the training error decreases satisfactorily. Then sensitivity matrices can be evaluated and the dimension of the input layer possibly reduced. Learning can subsequently be resumed until the training error reduces to an acceptable low value. This process can be repeated, however, usually only the first execution yields significant improvement. Numerical experiments indicate that the effort of additional network retraining can be too high in comparison to benefits of further minimization.

Should the redundancy in training data vectors exist, the proposed approach based on the average sensitivity matrices for input data pruning allows for more efficient training. This can be achieved at a relatively low computational cost and based on heuristic data pruning criteria outlined in the paper. The approach can be combined with other improved training strategies such as increased complexity training (Cloete and Ludik, 1993). Extension of the proposed sensitivity-based input pruning concept beyond continuous output values seems desirable for the case of networks with binary outputs such as classifiers and other binary encoders.

8. Minimum Sampling Rate for Information Encoding

An analytical approach allowing finding the optimum number of training examples for MLPNN can be outlined based on the Nyquist Sampling Theorem. The theorem states that *a function* $f(x)$ *which contains no frequency components greater than* f_0 *Hz is uniquely determined by the values of* $f(x)$ *at any set of sampling points spaced at most* $1/(2f_0)$ *seconds apart (Philips, 1990; Poularikas, 1992).* Sampling rates defined for time signals can be extended to other independent variables so that the generalized theorem for function approximation can be obtained. Each dimension of the transform will then correspond to one dimension of the original domain.

Obviously, sampling with a certain minimum frequency is needed to restore the signal from the samples taken. However, the theorem refers to the ideal case where the input signal has a finite high frequency boundary so that it can be accurately restored from samples using the inverse Fourier transform. Real-life signals are not band-limited, and we focus on the analysis of approximation conditions for MLPNNs.

An MLPNN trained using backpropagation algorithm can be regarded as a mean square error-based function approximation. However, no closed-form formulae for approximation polynomials are known before training. This makes it impossible to evaluate the sampling step theoretically. Therefore, the only hint for choosing the sufficient sampling interval is that it should be shorter than the Nyquist interval. The developed algorithm for non-band-limited functions is based on the assumption that only a certain fraction of information about the function is necessary for the approximation with a required accuracy.

Below we discuss the continuous function case. Let a continuous function to be approximated be known as $f(\boldsymbol{x})$,

$$f(\boldsymbol{x}) : \mathbb{D} \rightarrow \mathbb{R}, \qquad \mathbb{D} \subset \mathbb{R}^N$$

$$\mathbb{D} \doteq (x_{MIN_1} \ldots x_{MAX_1}) \times (x_{MIN_2} \ldots x_{MAX_2}) \times \ldots \times (x_{MIN_N} \ldots x_{MAX_M}) \quad (30)$$

$$\boldsymbol{x} \doteq [x_1, x_2, \ldots, x_n]$$

and let B_i be the range of the i-th variable, x_i:

$$B_i \doteq x_{MAX_i} - x_{MIN_i} \quad (31)$$

The multidimensional Fourier transform of $f(\boldsymbol{x})$ can now be expressed as

$$F(\boldsymbol{\Omega}) \doteq \frac{1}{(2\pi j)^N} \int_{x_{MIN_1}}^{x_{MAX_1}} \int_{x_{MIN_2}}^{x_{MAX_2}} \ldots \int_{x_{MIN_N}}^{x_{MAX_N}} f(x_1, x_2, \ldots, x_N) e^{2\pi j x_1 \omega_1} e^{2\pi j x_2 \omega_2}$$

$$\ldots e^{2\pi j x_N \omega_N} \, \mathrm{d}x_1 \mathrm{d}x_2 \ldots \mathrm{d}x_N \quad (32)$$

where $\boldsymbol{\Omega} \doteq [\omega_1, \omega_2, \ldots, \omega_N]$ is a vector of frequencies.

The criterion for minimum sampling frequency estimation can be formulated in a number of ways. First, the basic formula for optimization should be defined as a norm evaluating the information density at particular frequencies. The norm as defined in (33) has a meaning of generalized energy density:

$$E_d(\boldsymbol{\Omega}) \doteq |F(\boldsymbol{\Omega})|^2 \quad (33)$$

We can also use function (34) for evaluating the amount of information enclosed in the frequency band $\boldsymbol{\Omega}$. In case of a multidimensional band-limited function, the energy $E(\boldsymbol{\Omega})$, can be computed by integrating the generalized energy density (33) in the frequency domain in spherical (Andrews *et al.*, 1970), or more precisely, ellipsoidal coordinates within an N-dimensional ellipsoid. For example, in the simplest case assuming that function F has isotropic properties in each dimension ($\omega = \omega_1 = \omega_2 = \ldots = \omega_N$), we obtain

$$E(\boldsymbol{\Omega}) \doteq \int_0^{2\pi} \int_0^{2\pi} \ldots \int_0^{2\pi} \int_0^{1} |F(\omega r \cos\phi_1 \cos\phi_2 \ldots \cos\phi_{N-1}, \, \omega r \sin\phi_1 \cos\phi_2 \ldots \cos\phi_{N-1},$$

$$\ldots, \omega r \sin\phi_1 \sin\phi_2 \ldots \sin\phi_{N-1})|^2 r J(r, \phi_1, \phi_2, \ldots, \phi_N) \, \mathrm{d}\phi_1 \mathrm{d}\phi_2 \ldots \mathrm{d}\phi_{N-1} \mathrm{d}r \quad (34)$$

where $J(r, \phi_1, \phi_2, \ldots, \phi_N)$ is a term resulting from the change of the integration coordinates from cubic to spheric (Apostol, 1957). However, in general it cannot be assumed that the approximated function will have isotropic properties. It may then be

reasonable to choose smaller sampling densities in some dimension. The function (34) becomes then more complex due to the different boundaries in each dimension

$$E(\boldsymbol{\Omega}) \doteq \int_0^{2\pi}\int_0^{2\pi}\cdots\int_0^{2\pi}\int_0^{1} |F(\omega_1 r\cos\phi_1\cos\phi_2\ldots\cos\phi_{N-1},\ \omega_2 r\sin\phi_1\cos\phi_2\ldots\cos\phi_{N-1},$$

$$\ldots,\omega_N r\sin\phi_1\sin\phi_2\ldots\sin\phi_{N-1})|^2 J(r,\Omega,\phi_1,\phi_2,\ldots,\phi_N)\,\mathrm{d}\phi_1\mathrm{d}\phi_2\ldots\mathrm{d}\phi_{N-1}\mathrm{d}r \quad (35)$$

where $J(r,\boldsymbol{\Omega},\phi_1,\phi_2,\ldots,\phi_N)$ is a term obtained as previously after conversion of the integration coordinates from cubic to spheric.

Let C_{INFO} called the *information rate factor* be the fraction describing the required minimum energy content of the signal sampled with frequency $\boldsymbol{\Omega}$, normalized with respect to the total energy E_{TOT} of the original function (or function sampled with very high frequency). The information rate factor C_{INFO} can be seen as a theoretical measure of the information amount needed to approximate a function with a required accuracy.

Let function $f(\boldsymbol{x})$ be sampled with frequency $\boldsymbol{\Omega}$ satisfying the following condition:

$$\frac{E(\boldsymbol{\Omega})}{E(\boldsymbol{\Omega}_{MAX})} \geq C_{INFO} \quad (36)$$

where the frequency $\boldsymbol{\Omega}_{MAX} \doteq [\omega_1,\omega_2,\ldots,\omega_N]_{MAX}$ is high enough, so that

$$\frac{E(\boldsymbol{\Omega}_{MAX}) - E(\boldsymbol{\Omega})}{E(\boldsymbol{\Omega}_{MAX})} \ll C_{INFO} \quad (37)$$

Let us now compute the total number of samples in the training set. The number of samples taken per dimension M_{L_i}, is equal to

$$M_{L_i}(\omega_i) = |2B_i\omega_i + 1| \quad (38)$$

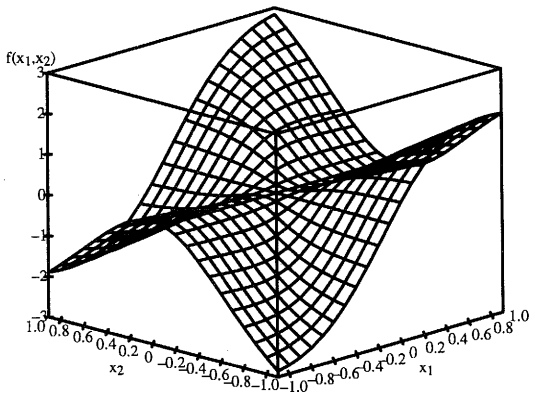
The total number of samples M_L can be expressed as

$$M_L(\boldsymbol{\Omega}) = \prod_{i=1}^{N} M_{L_i}(\omega_i) \quad (39)$$

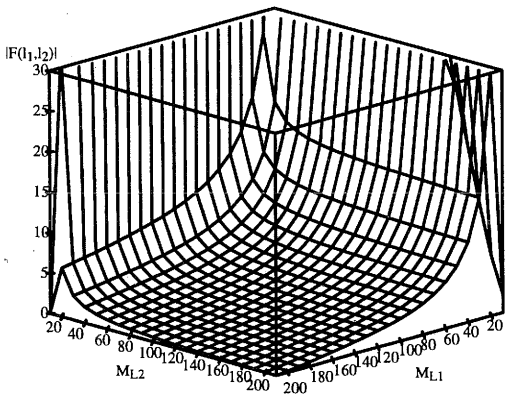
Our objective is to search among vectors $\boldsymbol{\Omega}$ which satisfy the condition (36) and to minimize the value of M_L defined by (39). The vector $\boldsymbol{\Omega}_{OPT} \doteq [\omega_1,\omega_2,\ldots,\omega_N]_{OPT}$ which is the solution to the given optimization problem contains the minimum sufficient sampling frequencies in the new training data set. The final sampling interval, Δx_i, is expected to be of different value for each dimension depending on the chosen frequency ω_i:

$$\Delta x_i = \frac{1}{2\omega_{OPT_i}} \quad (40)$$

Let us redefine the results for continuous functions for discrete data sets. Let us consider sampling a plant characteristic for which no closed-form formula exists.



(a)



(b)

Fig. 9. Approximated function $f(x)$ as in (53) (a), and its Fourier transform (b); ($P_{TOT} = 1.12$).

The average power P_{TOT} was calculated for the given function using formula (55); it is of value $P_{TOT} = 1.12$. MLPNN with 2 inputs, and 20 hidden neurons was trained to the error $MSE = 0.08$. Ψ and C_{INFO} were then calculated from equations (55) and (57) for $MSE = 0.08$ as $\Psi = 0.07$, and $C_{INFO} = 0.93$. The optimum number of samples for each dimension has been found from Fig. 11 by finding the minimum of M_L over frequencies satisfying condition (52) which shows the contour line bounding the domain of solution. This figure shows the contours for the number of samples in the training set M_L, for different l_1 and l_2 which satisfy condition (52). The minimum of M_L can be found at $L_1 = 4$ and $L_2 = 8$. It can be evaluated from equation (48). This corresponds to $M_{L_1} = 9$ samples for variable x_1 and $M_{L_2} = 17$ samples for variable x_2.

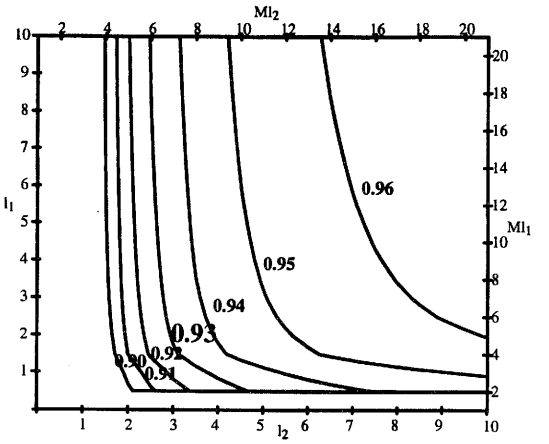


Fig. 10. Amount of energy $G_{NORM}(l)$ covered by bounded frequency spectrum ($E_{NORM} > 0.90$).

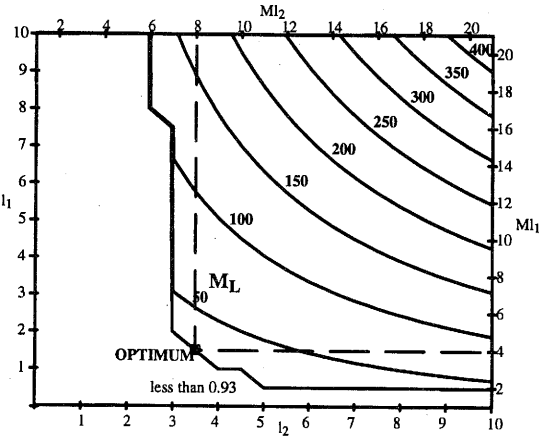
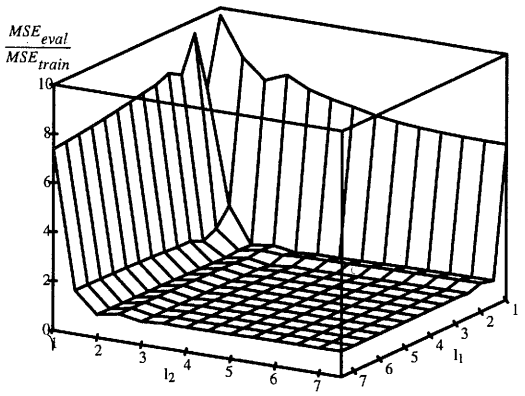
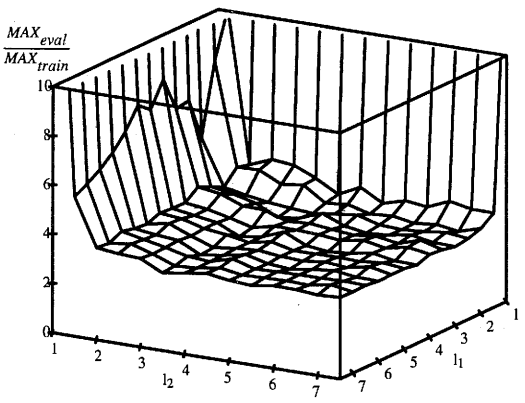


Fig. 11. Number of samples in learning set in area of normalized $G(l_1, l_2) > C_{INFO}$ ($C_{INFO} \geq 0.93$).

MLPNNs with architectures described above were then trained for different numbers of samples taken in each dimension to verify the theoretical results. The results are illustrated in Figs. 12–13. Figure 12 shows the quality of training in terms of MSE and the maximum error of approximation verification based on a very large test set (500×500 samples). It can be seen that the error decreases dramatically when $l_1 \geq 2$ and $l_2 \geq 3$. This corresponds to $M_{L_1} = 5$ and $M_{L_2} = 7$ samples per dimension, respectively.



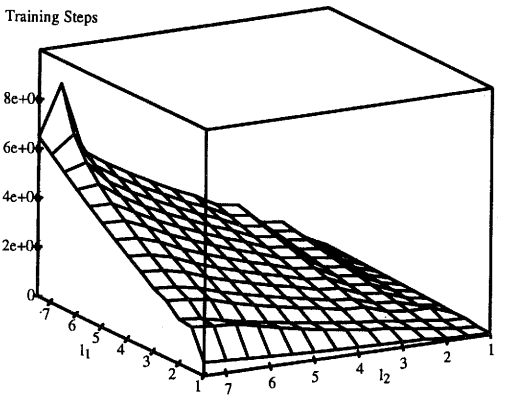
(a)



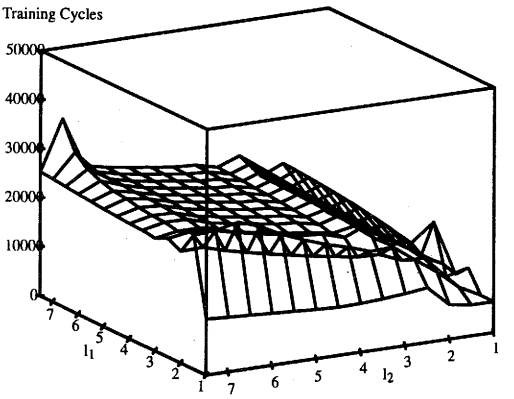
(b)

Fig. 12. Neural network performance (a) MSE (b) MAX.

Figure 13(a) shows the number of training steps required for the learning process, while Fig. 13(b) shows only the number of training cycles. After achieving certain frequencies of sampling the function to build a training set, the number of training cycles does not increase or increases only slowly, while the overall number of training steps still increases due to the growing number of training vectors.



(a)



(b)

Fig. 13. Number of training steps (a) and training cycles (b) versus sampling frequencies.

Figure 14 summarizes the computational experiment. It illustrates the number of training steps for the sampling frequencies $\boldsymbol{l}$ providing accurate learning. Local minima can be found for the frequencies $l_1 = 2$ and $l_2 = 5$. This corresponds to $M_{L_1} = 5$ and $M_{L_2} = 11$ samples per dimension, respectively. This is in agreement with $M_{L_1} = 4$ and $M_{L_2} = 8$ samples per appropriate dimension obtained by inspection of Fig. 11. The results are then rather close to those evaluated previously using the theoretical algorithm and shown in Fig. 11.

Both the results of the computational experiments and of theoretical studies show that the generalized sampling theorem can be applied to the approximation problem using neural networks. The smallest possible, but still large enough for the sake of accuracy data set should be selected, and then other network parameters can be

found through training. Our results indicate that the smallest training sets with only some data from the large measured real-life data sets are required in order to obtain successfully trained neural networks capable of accurate approximation.

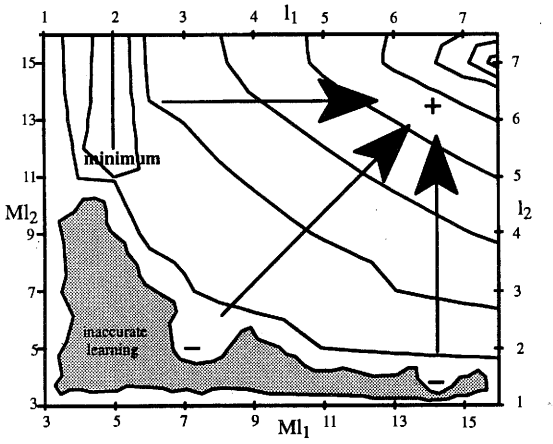


Fig. 14. Number of training steps versus sampling frequencies for area of satisfactory learning.

References

Andrews H.C., Pratt W.K. and Caspari K. (1970): *Computer Techniques in Image Processing.* — London: Academic Press Inc. Ltd.

Apostol T.M. (1957): *Mathematical Analysis. A Modern Approach to Advanced Calculus.* — London: Addison-Wesley Publishing Company, pp.270–275.

Cloete I. and Ludik J. (1993): *Increased complexity training.* — In: New Trends in Neural Computation, Eds. Mira J., Cabestany j. and Prieto A., in series Lecture Notes in Computer Science, Berlin: Springer-Verlag, pp.267–271.

Fu L. and Chen T. (1993): *Sensitivity analysis for input vector in multilayer feedforward neural networks.* — Proc. IEEE Int. Conf. *Neural Networks*, San Francisco (USA), March 28–April 1, pp.215–218.

Hartman E.J., Keeler J.D. and Kowalski J.M. (1990): *Layered neural networks with gaussian hidden units as universal approximators.* — Neural Computation, v.2, No.2, pp.210–215.

Kruschke J.K. and Movellan J.R. (1991): *Benefits of gain: speeded learning and minimal hidden layers in back-propagation networks.* — IEEE Trans. Systems, Man, and Cybernetics, v.21, No.1, pp.273–279.

Malinowski A., Żurada J.M. and Aronhime P.B. (1994): *Minimal training set size estimation for neural network-based function approximation.* — Proc. IEEE Int. Symp. *Circuits and Systems*, London (U.K.), May 28–June 2, pp.403–406.

Movellan J.R. (1987): *Self-regulated temperature in back-propagation networks.* — Presented at the Ninth Annual Berkeley-Stanford Conference, Berkeley, California.

Philips C.L. and Nagle H.T. (1990): *Digital Control Systems — Analysis and Design.* — New Jersey: Prentice Hall Inc.

Poularikas A.D. and Seely S. (1992): *Elements of Signals and Systems.* — Boston: PWS-KENT Publishing Company.

Shekhar S. and Amin M.B. (1992): *Generalization by neural networks.* — IEEE Trans. Knowledge and Data Eng., v.4, No.2, pp.177–185.

Tawel R. (1989): *Does the neuron 'learn' like synapse.* — In: Advances in Neural Information Processing Systems (Touretzky D.S., Ed.). – San Mateo (USA): Morgan Kaufmann, pp.169–176.

Żurada J.M. (1992): *Introduction to Artificial Neural Systems.* — St. Paul, (Minn.), West Publishing Company.

Żurada J.M. (1993): *Lambda learning rule for feedforward neural networks.* — Proc. IEEE Int. Conf. *Neural Networks*, San Francisco (USA), March 28–April 1, pp.1808–1811.

Żurada J.M., Malinowski A. and Cloete I. (1993): *Sensitivity analysis for pruning of training data in feedforward neural networks.* — Proc. First Australian and New Zeland Conf. *Intelligent Information Systems*, Perth, Western Australia, December 1–3, pp.288–292.

Żurada J.M., Malinowski A. and Cloete I. (1994): *Sensitivity analysis for minimization of input data dimension for feedforward neural network.* — Proc. IEEE Int. Symp. *Circuits and Systems*, London (U.K), May 28–June 2, pp.447–450.