Reports of the Department of Mathematical Information Technology
Series C. Software Engineering and Computational Intelligence
No. C 1/2000

MLP-Network in a Layer-Wise Form: Derivations, Consequences and Applications to Weight Decay

Tommi Kärkkäinen

University of Jyväskylä
Department of Mathematical Information Technology
P.O. Box 35 (Agora)
FIN–40351 Jyväskylä
FINLAND
fax +358 14 260 2731
http://www.mit.jyu.fi/

ISBN 951-39-0833-X
ISSN 1456-4378

MLP-Network in a Layer-Wise Form: Derivations, Consequences and Applications to Weight Decay

Tommi Kärkkäinen*

October 13, 2000

Abstract

Description of a feedforward MLP-network in a layer-wise algebraic form is given. A general calculus for the sensitivity analysis of transformations having the network-like nonlinear structure is developed. Based on the derivations some consequences of the least-means-squares learning problem for the locally optimal MLP are stated and further discussed. Numerical experiments with a presentation and comparison of different weight decay techniques are included.

Key words: MLP-network, layer-wise description, weight decay

1 Introduction

In this paper, we first consider the transformation that is realized by the feedforward neural network, i.e. MLP-network. Instead of the usual neuron-wise treatment of the network action with single computing units we consider a layer-wise description in an abstract setting using a linear-algebraic form based on matrices and function-matrices. In this way, we are able to construct a solid mathematical basis for earlier such presentations, e.g., in [26, 64]. The proposed formalism also suggests some interesting possibilities for generalizing the MLP-architecture.

An essential part of any textbook on neural networks consists of the derivation of error-backpropagation formulas for the network training [8, 63, 64]. What makes such sensitivity analysis messy is the consecutive application of the chain-rule in an index-jungle! In order to simplify and clarify this process, we develop a simple calculus for the sensitivity analysis of transformations having the layer-wise nonlinear structure of the MLP. The difficulties of the conventional approach are circumvented by applying consistently the Lagrangian treatment of equality constraint optimization problems [6]. Using the proposed calculus the derivation of the necessary optimality conditions for the least-means-squares (LMS) learning problem of MLP follows immediately. One advantage of the layer-wise formalism is that the optimality system is presented in a compact form that can be readily exploited in an efficient computer realization. In addition, due to the clear description of the optimality conditions we are able

*University of Jyväskylä, Department of Mathematical Information Technology, P.O.Box 35 (Agora), FIN-40351 Jyväskylä, Finland (tka@mit.jyu.fi)

to derive some interpretations and consequences concerning the final structure of the trained network. These results have direct applications to different weight decay techniques, which are presented and tested through numerical experiments.

One purpose of the current work is a thorough discussion of some of the common techniques used together with MLPs and other neural networks. Our point of view is based on the theory [6] and practice [58] of optimization which is used in order to enlighten some aspects originating from the LMS learning problem for the MLP training. The main emphasis in this respect is on the balance of unknowns in the optimization problem, which has direct consequences in preprocessing and weight decay.

The contents of the work are the following: First, in Section 2, we introduce the algebraic formalism of the multilayer feedforward network that is used throughout the rest of the paper. In Section 3, we derive the optimality conditions for a network learning in a similar form used to describe the network action. Also some consequences of the optimality conditions are described and discussed. Finally, in Section 4 we present numerical experiments for studying different regularization techniques and make some observations based on the computational results.

2 Layer-wise form of MLP-network

Next we develop step-by-step a layer-wise structure for the MLP-network. This will be accomplished by using an algebraic representation utilizing matrices and function-matrices [64, 26].

Step 1:

We start from the very beginning, i.e. from the *linear transformation* of given reals $x_1, \ldots, x_n$ that for $n = 1$ defines a straight line, for $n = 2$ a plane and for $n = 3, 4, \ldots$ a hyperplane.

$$a(\mathbf{x}) = w_0 + w_1 x_1 + \cdots + w_{n-1} x_{n-1} + w_n x_n = \mathbf{w}^T \hat{\mathbf{x}}, \tag{1}$$

where

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} \quad \text{and} \quad \hat{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}. \tag{2}$$

Here the special weight w_0 is called as *bias* node, and its purpose is to guarantee that the transformation does not necessarily go through the origin ($a(0) = w_0$). The extension of the original vector $\mathbf{x}$ into $\hat{\mathbf{x}}$ enables the compact notation $\mathbf{w}^T \hat{\mathbf{x}}$. The linear transformation is uniquely determined by the weight-vector $\mathbf{w}$. Especially, if $\|\mathbf{w}\| = 1$, then the real $a(\mathbf{x}) = \mathbf{w}^T \hat{\mathbf{x}}$ gives the length of the projection $p_{\mathbf{w}}(\hat{\mathbf{x}}) = \mathbf{w}^T \hat{\mathbf{x}}\, \mathbf{w}$ of $\hat{\mathbf{x}}$ onto $\mathbf{w}$.

Step 2:

We apply multiple linear transformations to the same vector $\mathbf{x}$ to obtain:

$$
\begin{aligned}
a_1(\mathbf{x}) &= w_{10} + w_{11}x_1 + \cdots + w_{1,n-1}x_{n-1} + w_{1,n}x_n = \mathbf{w}_1^T\hat{\mathbf{x}},\\
a_2(\mathbf{x}) &= w_{20} + w_{21}x_1 + \cdots + w_{2,n-1}x_{n-1} + w_{2,n}x_n = \mathbf{w}_2^T\hat{\mathbf{x}},\\
&\vdots\\
a_i(\mathbf{x}) &= w_{i,0} + w_{i,1}x_1 + \cdots + w_{i,n-1}x_{n-1} + w_{i,n}x_n = \mathbf{w}_i^T\hat{\mathbf{x}},\\
&\vdots\\
a_m(\mathbf{x}) &= w_{m,0} + w_{m,1}x_1 + \cdots + w_{m,n-1}x_{n-1} + w_{m,n}x_n = \mathbf{w}_m^T\hat{\mathbf{x}}.
\end{aligned}
\tag{3}
$$

By setting the weight-vectors $\mathbf{w}_1, \ldots, \mathbf{w}_m$ as rows (in, e.g., [64] columns are occupied and a transposed form in (5) is then obtained) of the matrix $\mathbf{W}$,

$$
\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_m^T \end{bmatrix}, \tag{4}
$$

we can present the m linear transformations in (3) in a compact form

$$
\mathbf{a} = \mathbf{W}\hat{\mathbf{x}}, \tag{5}
$$

where the vector $\mathbf{a}$ contains the obtained reals $a_1, \ldots, a_m$. This defines a two-layered structure with the input-layer represented by vector $\mathbf{x}$ and output-layer by $\mathbf{a}$.

Step 3:

Next, we add a nonlinear component to the above transformation. The usual way is to apply some suitable nonlinear function to each component of vector $\mathbf{a} = \mathbf{W}\hat{\mathbf{x}}$ to obtain $f(a_i) = f_i(a_i)$. This kind of a nonlinear transformation can be represented in a compact form (cf. [26]) by defining the so-called *diagonal function-matrix* $\mathcal{F} = \mathcal{F}(\cdot) = \mathrm{Diag}\{f_i(\cdot)\}_{i=1}^m$ as

$$
\mathcal{F} = \begin{bmatrix} f_1(\cdot) & 0 & \ldots & 0 \\ 0 & f_2(\cdot) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & f_m(\cdot) \end{bmatrix}. \tag{6}
$$

The function-matrix is supplied with natural way to define the matrix-vector product, i.e. $\mathbf{y} = \mathcal{F}(\mathbf{v}) \Leftrightarrow y_i = \sum_{j=1}^m f_{ij}(v_j)$ so that the multiplication of a vector component by a matrix component is simply replaced with an application of the corresponding function.
Using the above function-matrix definition we can consider the generalization of the linear transformation $\mathbf{a} = \mathbf{W}\hat{\mathbf{x}}$ into a form

$$
\mathcal{F}(\mathbf{a}) = \mathcal{F}(\mathbf{W}\hat{\mathbf{x}}). \tag{7}
$$

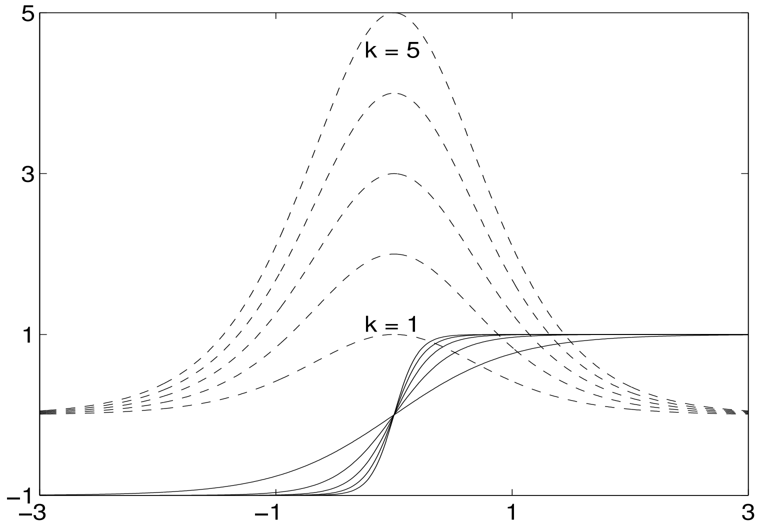

Figure 1: Functions $t_k(a)$ (solid line) and $t_k^{'}(a)$ (dashed line) for $k = 1, \ldots, 5$.

Here, the choice of function-matrix $\mathcal{F}$ as diagonal coincides with the usual way where each component a_i of vector $\mathbf{a}$ is transformed separately using the given activation function. Already at this point, we emphasize that in what follows all formulas are valid in the more general case where $\mathcal{F}$ is not diagonal (cf. Corollary 1).

For completeness, we include here the two most popular activation functions for MLP (although special activation functions for specific cases can be developed [59]). Typically used activation functions are bounded and monotone non-polynomials, so-called *squashing* functions [47, 63, 67, 84]. The most popular activation function is probably still the *logistic sigmoid* $1/(1 + \exp(-a))$ which is only a special case of the more general function ([21])

$$s_k(a) = \frac{1}{1 + \exp(-k\,a)}, \quad k = 1, 2, \ldots \tag{8}$$

that actually approaches the Rosenblatt's originally used step-function as the steepness parameter k goes to infinity. Notice that just for simplicity we have chosen k here to be a (positive) integer. The derivative of $s_k(a)$ reads as $s_k^{'}(a) = k\,\exp(k\,a)/(1 + \exp(k\,a))^2 = k\,s_k(a)\,(1 - s_k(a))$ and the inverse is given by $s_k^{-1}(a) = (\ln(a) - \ln(1-a))/k$. Another (generalization of) widely used activation function is the 'k-tanh' given by

$$t_k(a) = \frac{\exp(k\,a) - \exp(-k\,a)}{\exp(k\,a) + \exp(-k\,a)} = \frac{2}{1 + \exp(-2\,k\,a)} - 1 = 2\,s_k(2\,a) - 1, \tag{9}$$

with the derivative $t_k^{'}(a) = 4\,k\,\exp(2\,k\,a)/(1 + \exp(2\,k\,a))^2 = k\,(1 + t_k(a))\,(1 - t_k(a)) = k\,(1 - t_k(a)^2)$ and the inverse $t_k^{-1}(a) = (\ln(1+a) - \ln(1-a))/(2\,k)$. We remind that the given inverses are complex-valued for some arguments. The tanh-function and its derivatives are illustrated in Figure 1 for different values of k.

Notice that the two activation functions in (8) and (9) have different ranges. Moreover, for k fixed the 'k-tanh' grows from -1 to 1 twice as fast as the 'k-sigmoid' from 0 to 1 as can be seen from their closely related expressions and corresponding derivatives. This suggests that in

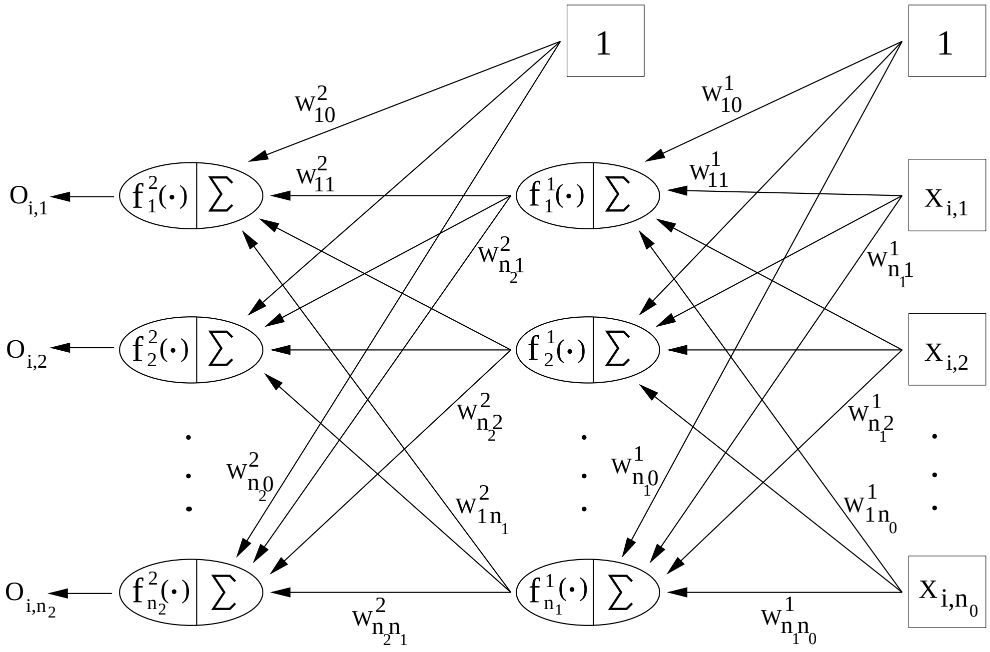

Figure 2: Perceptron with one hidden layer.

the basic form for $k = 1$ the 'k-tanh' function might be able to capture sharper boundaries, e.g., in classification. Anyway, smoothness of the nonlinear transformation containing the given activation functions can be controlled using the integer k.

Step 4:

However, usually only one nonlinear component in the feedforward transformation is not enough [32, 63]. For example, the perceptron as defined in (7) can correctly classify only data sets which are linearly separable. Thus, we need to introduce more than one subsequent nonlinear components (transformations), say two of them. Then we obtain the perceptron with one hidden layer in an *algebraic form* as

$$\mathbf{o} = \mathcal{N}(\mathbf{x}) = \mathcal{F}^2(\mathbf{W}^2 \widehat{\mathcal{F}}^1(\mathbf{W}^1 \hat{\mathbf{x}})). \tag{10}$$

Notice that we have placed the layer number (starting from zero for the input) as an upper index. Moreover, $\widehat{\mathcal{F}}^1$ means that the transformation of the hidden layer $\mathcal{F}^1(\mathbf{W}^1\hat{\mathbf{x}})$ must also be extended for obtaining the biases. As is well-known, by means of the universal approximation property of the MLP-networks, one hidden layer is already enough, if the number of nodes in it can grow to infinity [67, 84].
Finally, a compact notation for multilayer perceptron with $L-1$ hidden layers is given by

$$\mathbf{o} = \mathbf{o}^L = \mathcal{N}(\mathbf{x}) = \mathcal{F}^L(\mathbf{W}^L \hat{\mathbf{o}}^{(L-1)})), \tag{11}$$

where we have used the recursive definition

$$\begin{cases} \mathbf{o}^0 = \mathbf{x}, \\ \mathbf{o}^l = \mathcal{F}^l(\mathbf{W}^l \hat{\mathbf{o}}^{(l-1)}) \quad \text{for } l = 1, \dots, L. \end{cases}$$

The dimensions of the weight-matrices are given by $\dim(\mathbf{W}^l) = n_l \times (n_{l-1}+1),\ l = 1, \ldots, L$, where n_0 is the length of input-vectors $\{\mathbf{x}_i\}$, n_L the length of output-vectors $\{\mathbf{y}_i\}$, and $n_l, 0 < l < L$, determine the sizes (number of neurons) of hidden layers.

3 Layer-wise calculus for learning

3.1 MLP with one hidden layer

For clarity, we first consider the learning of perceptron with only one hidden layer which is illustrated in Figure 2.

3.1.1 Learning problem

The transformation realized by the one-hidden-layer perceptron is uniquely determined by the weight-matrices $\mathbf{W}^1 = (w^1_{ij})$ and $\mathbf{W}^2 = (w^2_{ij})$, and the chosen activation functions in $\mathcal{F}^1$ and $\mathcal{F}^2$. Because the activation functions are mostly fixed beforehand (also they could be optimized on some given set of admissible functions, e.g., w.r.t. k in (8) or (9)), we must be able to determine the values of w^1_{ij} and w^2_{ij} in order to activate the network.
This task is accomplished with training data, i.e. with a given set of desired input-output vector-pairs $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, $\mathbf{x}_i \in \mathbb{R}^{n_0}$ and $\mathbf{y}_i \in \mathbb{R}^{n_2}$, which are utilized to configure the network in such a way that $\mathcal{N}(\mathbf{x}_i) \sim \mathbf{y}_i$ for all $i = 1, \ldots, N$ in appropriate sense. For this purpose, we introduce a learning problem and the corresponding training algorithm for solving this problem.
A natural configuration principle for the network is the requirement that the average error of the mapping generated by the MLP over the learning data should be as small as possible. The mathematical formulation corresponding to this principle yields determination of $\mathbf{W}^1$ and $\mathbf{W}^2$ as a solution of the optimization problem

$$\min_{(\mathbf{W}^1, \mathbf{W}^2)} \mathcal{J}(\mathbf{W}^1,\, \mathbf{W}^2), \tag{12}$$

where

$$\mathcal{J}(\mathbf{W}^1,\, \mathbf{W}^2) = \frac{1}{2N}\sum_{i=1}^N \|\mathcal{N}(\mathbf{x}_i) - \mathbf{y}_i\|^2 = \frac{1}{2N}\sum_{i=1}^N \|\mathcal{F}^2(\mathbf{W}^2\, \widehat{\mathcal{F}}^1(\mathbf{W}^1\, \hat{\mathbf{x}}_i)) - \mathbf{y}_i\|^2 \tag{13}$$

is the least-mean-squares (LMS) cost functional in its basic form. Here $\|\cdot\|$ denotes the l^2-norm of vector which is induced by the corresponding inner product $(\mathbf{v}, \mathbf{w}) = \mathbf{w}^T\mathbf{v}$.
One immediately notices that all outputs $\mathcal{N}(\mathbf{x}_i)$ in (13) are restricted to the range of the activation functions in $\mathcal{F}^2$. Therefore, one should preprocess at least the given set of desired outputs $\{\mathbf{y}_i\}$ into this range. Moreover, when considering gradient-descent based training, one must try to keep all unknowns in $\mathbf{W}^1$ and $\mathbf{W}^2$ on the same level. Otherwise, components of the gradient have different orders of magnitude, which may yield nonbalanced updates in the training algorithm. Finally, this usually leads to large deviation of weights decreasing the fault tolerance of a network substantially [11]. Therefore, we suggest that all activation functions in $\mathcal{F}^1$ and $\mathcal{F}^2$ have the same range and also the input-data $\{\mathbf{x}_i\}$ is prescaled into

this range. Then all layers of the network treat vectors with components of the same order of magnitude.
To this end, we here base the network learning on the following simplified form of (13):

$$\mathcal{J}(\mathbf{W}^1, \mathbf{W}^2) = \frac{1}{2N} \sum_{i=1}^{N} \|\mathbf{W}^2 \, \widehat{\mathcal{F}}(\mathbf{W}^1 \hat{\mathbf{x}}_i) - \mathbf{y}_i\|^2, \tag{14}$$

where the learning data $\{\mathbf{x}_i, \mathbf{y}_i\}$ is assumed to be prescaled onto the interval $[0, 1]$ (k-sigmoidal activation) or $[-1, 1]$ (k-tanh activation). The relation between (13) and (14) will be discussed in Section 3.3 in Remark 2. Notice that a possible loss of information using k-tanh activation and the proposed prescaling is that for each feature in $\{\mathbf{x}_i\}$ the average of minimum and maximum values is transformed to zero. Hence, all records containing this average value are insensitive to the corresponding column in the weight-matrix $\mathbf{W}^1$ (multiplication of zero). Hence, this may result to non-uniqueness in $\mathbf{W}^1$ and lead to a substantial loss of information especially for those features having independent Gaussian distribution.
Such least-squares formulations are still nowadays the most popular ones for training the MLP, but also other formulations based on robust statistics exist [31]. In case of noisy data one can alter the underlying assumption behind least-squares on having Gaussian data-error distribution. This directly leads to a derivation of new cost functionals for the learning problem (e.g., [13, 48]).

3.1.2 Optimality conditions in algebraic form

For a gradient-based training algorithm we need expressions for the derivatives $\frac{\partial \mathcal{J}}{\partial w^1_{ij}}$ and $\frac{\partial \mathcal{J}}{\partial w^2_{ij}}$. Namely, a local solution $(\mathbf{W}^{1^*}, \mathbf{W}^{2^*})$ of the minimization problem (12) is characterized by the conditions

$$\nabla_{(\mathbf{W}^1, \mathbf{W}^2)} \mathcal{J}(\mathbf{W}^{1^*}, \mathbf{W}^{2^*}) = \begin{bmatrix} \nabla_{\mathbf{W}^1} \mathcal{J}(\mathbf{W}^{1^*}, \mathbf{W}^{2^*}) \\ \nabla_{\mathbf{W}^2} \mathcal{J}(\mathbf{W}^{1^*}, \mathbf{W}^{2^*}) \end{bmatrix} = \begin{bmatrix} \mathbf{O} \\ \mathbf{O} \end{bmatrix}. \tag{15}$$

Here, $\nabla_{\mathbf{W}^l} \mathcal{J} = \left[\frac{\partial \mathcal{J}}{\partial w^l_{ij}} \right]_{i,j}$, $l = 1, 2$, which due to the applied formalism is presented in a similar matrix-form as the unknown weight-matrices. Hence, our next task is to derive the partial derivatives w.r.t. these 'data structures' (cf. [19] Appendix B and [26]). For this analysis, we presuppose that all activation functions in the function-matrix $\mathcal{F}$ are differentiable.
We start the derivation by giving some simple, but useful lemmas and one corollary for which the proofs are contained in Appendix A. We remind that the proposed approach is not restricted to the LMS error function. For other (differentiable) cost functionals one can use exactly the same technique for deriving the necessary optimality conditions in a similar layer-wise form.

Lemma 1. Let $\mathbf{v} \in \mathbb{R}^{m_1}$ and $\mathbf{y} \in \mathbb{R}^{m_2}$ be given vectors. The gradient-matrix $\nabla_{\mathbf{W}} J(\mathbf{W}) \in \mathbb{R}^{m_2 \times m_1}$ for the functional

$$J(\mathbf{W}) = \frac{1}{2} \|\mathbf{W}\,\mathbf{v} - \mathbf{y}\|^2$$

is of the form

$$\nabla_{\mathbf{W}} J(\mathbf{W}) = [\mathbf{W}\,\mathbf{v} - \mathbf{y}]\, \mathbf{v}^T.$$

Lemma 2. Let $\mathbf{W} \in \mathbb{R}^{m_2 \times m_1}$ be a given matrix, $\mathbf{y} \in \mathbb{R}^{m_2}$ a given vector and $\mathcal{F} = \mathrm{Diag}\{f_i\}_{i=1}^{m_1}$ a given, diagonal function-matrix. The gradient-vector $\nabla_{\mathbf{u}} J(\mathbf{u}) \in \mathbb{R}^{m_1}$ for the functional

$$J(\mathbf{u}) = \frac{1}{2} \, \|\mathbf{W} \, \mathcal{F}(\mathbf{u}) - \mathbf{y}\|^2 \tag{16}$$

reads as

$$\nabla_{\mathbf{u}} J(\mathbf{u}) = \mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{u})\} \, \mathbf{W}^T \, [\mathbf{W} \, \mathcal{F}(\mathbf{u}) - \mathbf{y}].$$

Corollary 1. If in Lemma 2 the function-matrix $\mathcal{F}(u)$ is of a general form (not necessarily diagonal), the gradient-vector $\nabla_{\mathbf{u}} J(\mathbf{u}) \in \mathbb{R}^{m_1}$ for the functional (16) has the presentation

$$\nabla_{\mathbf{u}} J(\mathbf{u}) = \left(\mathbf{W} \, \mathcal{F}^{'}(\mathbf{u})\right)^T \, [\mathbf{W} \, \mathcal{F}(\mathbf{u}) - \mathbf{y}],$$

where $\mathcal{F}^{'}(\mathbf{u})$ denotes the function-matrix-multiplication of vector $\mathbf{u}$ by the derivative functions.

Notice that the result in Corollary 1 is actually a natural generalization of the optimality condition $\mathbf{A}^T(\mathbf{A}\,\mathbf{u} - \mathbf{y}) = 0$ for the quadratic problem

$$\min_{u \in \mathbb{R}^n} \frac{1}{2} \, \|\mathbf{A}\,\mathbf{u} - \mathbf{y}\|^2 = \frac{1}{2} \, (\mathbf{A}\,\mathbf{u} - \mathbf{y})^T \, (\mathbf{A}\,\mathbf{u} - \mathbf{y}).$$

The next lemma contains the fundamental result for deriving the optimality conditions of the three-layered perceptron in algebraic form. Its proof in Appendix A also introduces the Lagrangian technique that has been used here for obtaining such results.

Lemma 3. Let $\bar{\mathbf{W}} \in \mathbb{R}^{m_2 \times m_1}$ be a given matrix, $\mathcal{F} = \mathrm{Diag}\{f_i\}_{i=1}^{m_1}$ a given, diagonal function-matrix, and $\mathbf{v} \in \mathbb{R}^{m_0}$, $\mathbf{y} \in \mathbb{R}^{m_2}$ given vectors. The gradient-matrix $\nabla_{\mathbf{W}} J(\mathbf{W}) \in \mathbb{R}^{m_1 \times m_0}$ for the functional

$$J(\mathbf{W}) = \frac{1}{2} \, \|\bar{\mathbf{W}} \, \mathcal{F}(\mathbf{W}\mathbf{v}) - \mathbf{y}\|^2 \tag{17}$$

is of the form

$$\nabla_{\mathbf{W}} J(\mathbf{W}) = \mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{W}\mathbf{v})\} \, \bar{\mathbf{W}}^T \, [\bar{\mathbf{W}} \, \mathcal{F}(\mathbf{W}\mathbf{v}) - \mathbf{y}] \, \mathbf{v}^T.$$

Now we are ready to give the actual result for the perceptron with one hidden layer.

Theorem 1. Gradient-matrices $\nabla_{\mathbf{W}^2} \mathcal{J}(\mathbf{W}^1, \mathbf{W}^2)$ and $\nabla_{\mathbf{W}^1} \mathcal{J}(\mathbf{W}^1, \mathbf{W}^2)$ for the cost functional (14) are of the form

(i)
$$\begin{aligned}\nabla_{\mathbf{W}^2} \mathcal{J}(\mathbf{W}^1, \mathbf{W}^2) &= \frac{1}{N} \sum_{i=1}^{N} [\mathbf{W}^2 \, \widehat{\mathcal{F}}(\mathbf{W}^1 \, \hat{\mathbf{x}}_i) - \mathbf{y}_i] \, [\widehat{\mathcal{F}}(\mathbf{W}^1 \, \hat{\mathbf{x}}_i)]^T \\ &= \frac{1}{N} \sum_{i=1}^{N} \mathbf{e}_i \, [\widehat{\mathcal{F}}(\mathbf{W}^1 \, \hat{\mathbf{x}}_i)]^T,\end{aligned}$$

(ii)
$$\begin{aligned}\nabla_{\mathbf{W}^1} \mathcal{J}(\mathbf{W}^1, \mathbf{W}^2) &= \frac{1}{N} \sum_{i=1}^{N} \mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{W}^1 \hat{\mathbf{x}}_i)\} \, (\mathbf{W}^2_1)^T \, [\mathbf{W}^2 \, \widehat{\mathcal{F}}(\mathbf{W}^1 \, \hat{\mathbf{x}}_i) - \mathbf{y}_i] \, \hat{\mathbf{x}}_i^T \\ &= \frac{1}{N} \sum_{i=1}^{N} \mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{W}^1 \, \hat{\mathbf{x}}_i)\} \, (\mathbf{W}^2_1)^T \, \mathbf{e}_i \, \hat{\mathbf{x}}_i^T.\end{aligned}$$

In (ii), $\mathbf{W}_1^2$ is the submatrix $(\mathbf{W}^2)_{i,j}$, $i = 1, \ldots, n_2$, $j = 1, \ldots, n_1$, which is obtained from $\mathbf{W}^2$ by removing the first column $\mathbf{W}_0^2$ containing the bias nodes.

Proof. Formula (i) is a direct consequence of Lemma 1. Moreover, due to the definition of the extension operator $\widehat{\ }$ we have, for all $1 \le i \le N$,

$$\mathbf{W}^2 \widehat{\mathcal{F}}(\mathbf{W}^1 \hat{\mathbf{x}}_i) - \mathbf{y}_i = [\mathbf{W}_0^2 \ \mathbf{W}_1^2] \begin{bmatrix} 1 \\ \mathcal{F}(\mathbf{W}^1 \hat{\mathbf{x}}_i) \end{bmatrix} - \mathbf{y}_i = \mathbf{W}_0^2 + \mathbf{W}_1^2 \mathcal{F}(\mathbf{W}^1 \hat{\mathbf{x}}_i) - \mathbf{y}_i. \tag{18}$$

Using (18) and Lemma 3 componentwise with the cost functional (14) shows (ii) and ends the proof. □

Notice that the optimality conditions above are given in the form suitable for individual activation functions represented as a diagonal function-matrix. As stated in Corollary 1, similar conditions remain valid in case of a more general, not necessarily diagonal function-matrix activation. From the training point of view, this opens up new possibilities for generalizing the basic architecture of MLP (remember that the re-invention of MLP during 80's was due to backpropagation, a way to train the architecture), but this topic is out of the scope of this work. A first step towards this direction should clearly be a study of the universal approximation properties of such generalized architectures. Finally, other generalization techniques can yield, e.g., recurrent networks [63] and networks with hysteresis [17].

3.2 Several hidden layers

Next, we generalize the previous analysis to the case of several hidden layers. The cost functional to be minimized is now given by

$$\mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L) = \frac{1}{2N} \sum_{i=1}^N \|\mathbf{W}^L \hat{\mathbf{o}}_i^{(L-1)} - \mathbf{y}_i\|^2, \tag{19}$$

where $\mathbf{o}_i^0 = \mathbf{x}_i$ and $\mathbf{o}_i^l = \mathcal{F}^l(\mathbf{W}^l \hat{\mathbf{o}}_i^{(l-1)})$ for $l = 1, \ldots, L-1$.
The next lemma (which is also proved in Appendix A) generalizes the result of Lemma 3 to produce a fundamental result for deriving the optimality conditions in an algebraic form for an MLP with more than one hidden layer.

Lemma 4. Let $\tilde{\mathbf{W}} \in \mathbb{R}^{m_3 \times m_2}$ and $\bar{\mathbf{W}} \in \mathbb{R}^{m_2 \times m_1}$ be given matrices, $\tilde{\mathcal{F}} = \mathrm{Diag}\{f_i\}_{i=1}^{m_2}$ and $\bar{\mathcal{F}} = \mathrm{Diag}\{f_i\}_{i=1}^{m_1}$ given, diagonal function-matrices, and $\mathbf{v} \in \mathbb{R}^{m_0}$, $\mathbf{y} \in \mathbb{R}^{m_3}$ given vectors. The gradient-matrix $\nabla_{\mathbf{W}} J(\mathbf{W}) \in \mathbb{R}^{m_1 \times m_0}$ for the functional

$$J(\mathbf{W}) = \frac{1}{2} \|\bar{\mathbf{W}} \bar{\mathcal{F}}(\tilde{\mathbf{W}} \tilde{\mathcal{F}}(\mathbf{W}\mathbf{v})) - \mathbf{y}\|^2 \tag{20}$$

is of the form

$$\nabla_{\mathbf{W}} J(\mathbf{W}) = \mathrm{Diag}\{\tilde{\mathcal{F}}'(\mathbf{W}\mathbf{v})\} \tilde{\mathbf{W}}^T \mathrm{Diag}\{\bar{\mathcal{F}}'(\tilde{\mathbf{W}} \tilde{\mathcal{F}}(\mathbf{W}\mathbf{v}))\} \bar{\mathbf{W}}^T [\bar{\mathbf{W}} \bar{\mathcal{F}}(\tilde{\mathbf{W}} \tilde{\mathcal{F}}(\mathbf{W}\mathbf{v})) - \mathbf{y}] \mathbf{v}^T.$$

Theorem 2. Gradient-matrices $\nabla_{\mathbf{W}^l}\ \mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L)$, $l = L, \ldots, 1$, for the cost functional (19) are of the form

$$\nabla_{\mathbf{W}^l}\mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L) = \frac{1}{N}\sum_{i=1}^N \mathbf{d}_i^l\,[\hat{\mathbf{o}}_i^{(l-1)}]^T,$$

where

$$\mathbf{d}_i^L = \mathbf{e}_i = \mathbf{W}^L\hat{\mathbf{o}}_i^{(L-1)} - \mathbf{y}_i, \tag{21}$$

$$\mathbf{d}_i^l = \mathrm{Diag}\{(\mathcal{F}^l)'(\mathbf{W}^l\,\hat{\mathbf{o}}_i^{(l-1)})\}\,(\mathbf{W}_1^{(l+1)})^T\,\mathbf{d}_i^{(l+1)}. \tag{22}$$

Proof. Apply Lemma 4 inductively. □

Remark 1. Few observations concerning Theorem 2.

(*i*) Definition of $\mathbf{d}_i^l$ in Theorem 2 contains the backpropagation of the output-error in a layer-wise form.

(*ii*) Theorem 2 gives formulas for the derivatives in a compact form that can be readily exploited in the implementation. Moreover, computation of the activation-derivatives $(\mathcal{F}^l)'(\mathbf{W}^l\,\hat{\mathbf{o}}_i^{(l-1)})$ can, using the proposed sigmoidal activation functions in (8) and (9), be realized using layer-outputs $\mathbf{o}_i^l$. Hence, when going through the network for the first time for obtaining the output $\mathcal{N}(x_i)$, it is enough to store the individual outputs of different layers for the backward gradient loop. For a minimal memory usage these same vectors can then be overwritten by $\mathbf{d}_i^l$ if in the actual implementation the whole operation in (22) is performed in a single loop. In modern workstations such combination of operations yielding minimal amount of loops through the memory can decrease the CPU time substantially, cf. [40].

3.3 Some consequences of the optimality conditions

Next we derive some straightforward consequences of the layer-wise optimality conditions. First result shows especially that *every* local solution $\{\mathbf{W}^{l^*}\}_{l=1}^L$ obtained by solving the least-squares optimization problem actually satisfies our initial intention $\mathcal{N}^*(\mathbf{x}_i) \sim \mathbf{y}_i$ in a particular statistical sense.

Corollary 2. (*i*) The average error $\frac{1}{N}\sum_{i=1}^N \mathbf{e}_i^*$ made by the locally optimal MLP-network satisfying the conditions in Theorem 2 is zero.

(*ii*) The correlation between the error-vectors and the action of layer $L-1$ is zero.

Proof. The optimality condition $\nabla_{\mathbf{W}^L}\mathcal{J} = \frac{1}{N}\sum_{i=1}^N \mathbf{e}_i^*\,[\hat{\mathbf{o}}_i^{(L-1)}]^T = \mathbf{O}$ (with the abbreviation $\hat{\mathbf{o}}_i^{(L-1)} = \hat{\mathbf{o}}_i^{(L-1)^*}$) in Theorem 2 can be written in the non-extended form as

$$\frac{1}{N}\sum_{i=1}^N \mathbf{e}_i^*\,[1\ (\mathbf{o}_i^{(L-1)})^T] = \mathbf{O}. \tag{23}$$

By taking the transpose in (23), we obtain

$$\frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix}1\\ \mathbf{o}_i^{(L-1)}\end{bmatrix}(\mathbf{e}_i^*)^T = \frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix}(\mathbf{e}_i^*)^T\\ \mathbf{o}_i^{(L-1)}(\mathbf{e}_i^*)^T\end{bmatrix} = \begin{bmatrix}\mathbf{O}\\ \mathbf{O}\end{bmatrix}. \tag{24}$$

But now the first row in (24) shows (i) and the second row (ii) thus ending the proof. □

Remark 2. **Final layer with or without activation?**

- From Lemma 3, it follows that if MLP also contains the final layer activation

  $$\mathcal{N}(\mathbf{x}_i) = \mathcal{F}^L(\mathbf{W}^L\hat{\mathbf{o}}_i^{(L-1)})),$$

  then (21) replaced with

  $$\mathbf{d}_i^L = \mathrm{Diag}\{(\mathcal{F}^L)'(\mathbf{W}^L\,\hat{\mathbf{o}}_i^{(L-1)})\}\,\mathbf{e}_i$$

  gives the corresponding sensitivity w.r.t. $\mathbf{W}^L$. In this case, we have in Corollary 2 instead of (24)

  $$\frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix}(\mathbf{e}_i^*)^T\\ \mathbf{o}_i^{(L-1)}(\mathbf{e}_i^*)^T\end{bmatrix}\mathbf{D}_i = \begin{bmatrix}\mathbf{O}\\ \mathbf{O}\end{bmatrix}$$

  for $\mathbf{D}_i = \mathrm{Diag}\{(\mathcal{F}^L)'(\mathbf{W}^L\,\hat{\mathbf{o}}_i^{(L-1)})\}$. Hence, the two formulations with and without activating the final layer yield locally the same result only for the zero-residual problem $\mathbf{e}_i^* = 0$ for all $1 \le i \le N$. This slightly generalizes the corresponding result derived in [55] where a bijectivity of the final activation $\mathcal{F}^L$ was also assumed.

- The use of 0–1 coding for the desired output-vectors in classification enforces the weight-vector of 1-neuron to the so-called saturation area ([80]) of a sigmoidal activation where the derivative is nearly zero (cf. Figure 1). For this reason, the error function with the final layer activation has a nearly flat region around such points. Indeed, the tests reported in [23] suggest that the network with sigmoid in the final layer has more local minima than when using a linear final layer. Furthermore, in [35] it has been pointed out that the reconstruction error surface for a sigmoidal autoassociator consists of multiple local valleys on the contrary to the linear one. Hence, one conclusion that has been drawn is that the nonlinear transformation acts locally whereas the linear one globally. By combining these two formulations in (19) we try to take advantage of both these characteristics.

- To have some idea on the complexity of the learning problems with and without the final layer activation, we have illustrated in Figure 3 two error functions in the two-dimensional weight-space (w^1, w^2) (cf. Figure 8.7 in [63]). More precisely, remembering the role of bias as shift of the transformation from the origin, we have computed the LMS error function for the two nonlinear mappings

  $\mathcal{N}_n(x) = t_1(w^2\,t_1(w^1x))$: nonlinear final layer as in (13),
  $\mathcal{N}_l(x) = w^2\,t_1(w^1x)$: linear final layer as in (14),

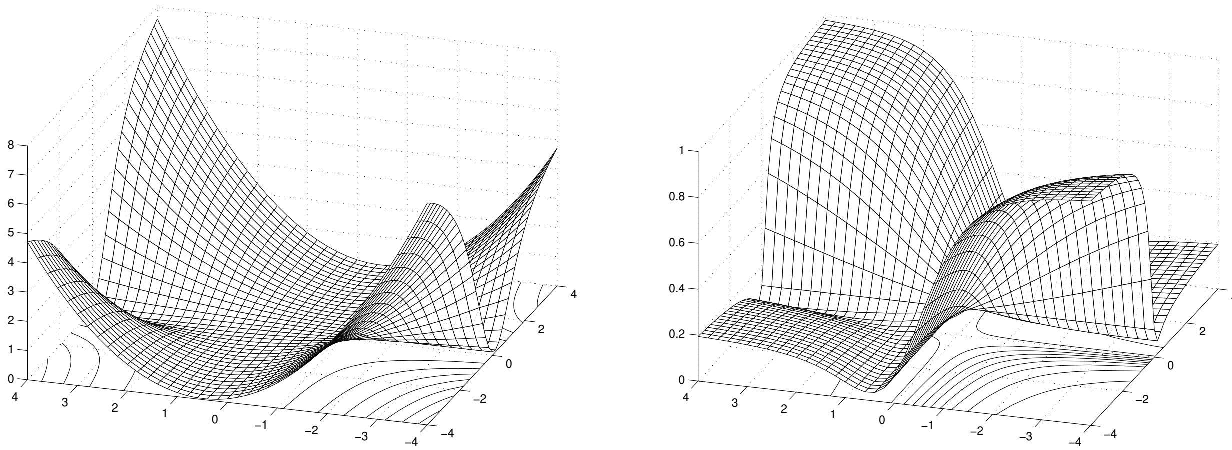

Figure 3: Error functions in the two-dimensional weight-space for linear (left) and nonlinear (right) final layers in MLP.

where the data $\{x_i, y_i\}$ is the prescaled noisy data representing the sine-function in Example 1 in Section 4. Figure 3 clearly shows the flatness problems of the error function resulting from the final layer activation. Furthermore, this also suggest existence of multiple local minima with hidden nonlinear transformations, especially, when combining more of them together [16].

In the next corollary we make a further discussion concerning the last weight-matrix $\mathbf{W}^L$.

Corollary 3. If the autocorrelation matrix $\mathbf{A} = \frac{1}{N}\sum_{i=1}^{N} \widehat{\mathbf{v}}_i \, \widehat{\mathbf{v}}_i^T$ corresponding to the (extended) final layer inputs $\mathbf{v}_i = \mathbf{o}_i^{(L-1)}$ is non-singular, $\mathbf{W}^L$ can be recovered from the formula

$$\mathbf{W}^L = \mathbf{B}\,\mathbf{A}^{-1} \quad \text{for} \;\; \mathbf{B} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_i \, \widehat{\mathbf{v}}_i^T. \tag{25}$$

Furthermore, if $\mathbf{A}^*$ for a minimizer $\{\mathbf{W}^{l^*}\}_{l=1}^{L}$ of (19) is non-singular, then $\mathbf{W}^{L^*}$ is unique.

Proof. We notice that $\mathbf{W}^L$ satisfying the system

$$\frac{1}{N}\sum_{i=1}^{N} [\mathbf{W}^L \, \hat{\mathbf{o}}_i^{(L-1)} - \tilde{\mathbf{y}}_i] \, [\hat{\mathbf{o}}_i^{(L-1)}]^T = \mathbf{O} \tag{26}$$

is independent of $i = 1, \ldots, N$. This means that the equation (26) can be written as

$$\mathbf{W}^L \frac{1}{N}\sum_{i=1}^{N} \widehat{\mathbf{v}}_i \, \widehat{\mathbf{v}}_i^T = \frac{1}{N}\sum_{i=1}^{N} \tilde{\mathbf{y}}_i \, \widehat{\mathbf{v}}_i^T \quad \Leftrightarrow \quad \mathbf{W}^L \, \mathbf{A} = \mathbf{B}. \tag{27}$$

Multiplying both sides in this equation from right with $\mathbf{A}^{-1}$ gives (25) and ends the proof. □

Let us add one further observation concerning the above result. Consider the three-layered perceptron with the autocorrelation matrix $\mathbf{A} = \frac{1}{N}\sum_{i=1}^{N} \widehat{\mathbf{v}}_i \, \widehat{\mathbf{v}}_i^T \in \mathbb{R}^{n_1+1 \times n_1+1}$. Because $\dim(\mathrm{span}(\widehat{\mathbf{v}}_i)) =$

1, each matrix $\widehat{\mathbf{v}}_i\, \widehat{\mathbf{v}}_i^T$ has only one nonzero eigenvalue $\lambda = \widehat{\mathbf{v}}_i^T\, \widehat{\mathbf{v}}_i$ with the corresponding eigenvector $\widehat{\mathbf{v}}_i$. This means that $1 \leq \dim(\mathrm{span}(\mathbf{A})) \leq \min(N, n_1 + 1)$. Hence, if $N < n_1 + 1$ the matrix $\mathbf{A}$ must be singular and $\mathbf{W}^{2^*}$ can not be unique. On the other hand, if $N > n_1 + 1$, all vectors $\widehat{\mathbf{v}}_i$ can not be linearly independent.
To this end, notice that in Corollary 3 the assumption on nonsingularity can be relaxed to the so-called *pseudo-invertibility*, which has been extensively used to generate new training algorithms for MLP-networks [12, 18, 82].

Remark 3.
- Due to the given layer-wise description of MLP formulas for the gradient-matrices can easily be checked using some software capable of symbolic calculations.
- The consequences that were stated here followed immediately from the optimality conditions w.r.t the final layer of MLP. It remains as a future challenge to analyse the conditions for hidden layers more profoundly in order to get insight into the way how MLP solves problems for us. We believe that a possible transition from the neuron-wise to the layer-wise treatment could be helpful for such investigations.

3.4 Comments and remarks

Next, we give a list of observations and comments based on the results in Section 3.3.

Role of Corollary 2: Notice that both results in Corollary 2 are consequences of including the bias-term in the input $\mathbf{o}_i^{(L-1)}$ of the *last, linear* layer $\mathbf{W}^L$. Especially, the transformation before the last layer has no influence whatsoever on these facts. This suggests the following interpretation of the MLP-action with the proposed architecture: nonlinear hidden layers produce the universal approximation capability of MLP while the final linear layer compensates the hidden action with the desired output in an uncorrelated and error-averaging manner (cf. [35]).

Assume that we have noise in the given output data:

$$\mathbf{y}_i = \bar{\mathbf{y}}_i + \varepsilon_i \quad 1 \leq i \leq N,$$

where $\bar{\mathbf{y}}_i$ denotes the error-free (true) output-vector. It then follows from Corollary 2, (i) that the output-error of the locally optimal MLP-network $\mathcal{N}^*$ satisfies

$$\frac{1}{N}\sum_{i=1}^{N} [\mathcal{N}^*(x_i) - \bar{\mathbf{y}}_i] = \frac{1}{N}\sum_{i=1}^{N} \varepsilon_i.$$

Hence, *noise with zero mean in the output-data is complete removed by the locally optimal MLP-network.* To analyse the case of noisy input-data $\{\mathbf{x}_i\}$ certainly requires more comprehensive study, because such errors are amplified by the transformation realized by the MLP-network. We remind that, e.g., in classification the class $\mathbf{y}_i$ of an individual observation $\mathbf{x}_i$ is usually known beforehand being thus error-free, but the features in $\mathbf{x}_i$ may contain all kinds of degradation.

It is known that any model trained using the LMS cost functional approximates the Bayes optimal discriminant function (BODF) in classification [65]. In this case, the architecture of MLP determining its flexibility should be related to the complexity of BODF. On the other hand, BODF does not necessarily minimize the probability of classification error [57], and the basic assumption that samples from different classes are independent is a severe restriction. Relaxation of such assumption yields other classifiers which can still be implemented using neural networks [81].

Simple modifications of LMS cost functional: Consider the classification problem with learning data from K different classes $\{C_k\}_{k=1}^K$ so that $N = \sum_{k=1}^K N_k$, where N_k denotes the number of samples from the class C_k. It then follows from Corollary 2, (i) that the amount of learning data N_k from an individual class has a significant effect to the obtained MLP-classifier (cf. the BODF-result above). Hence, if one would like to have equal probability $\frac{1}{K}$ for all classes in the classifier, the least-squares cost functional to be minimized should be adjusted for this purpose using the weighted form

$$\mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L) = \sum_{k=1}^K \frac{1}{2KN_k} \sum_{i \in C_k} \|\mathbf{W}^L \hat{\mathbf{o}}_i^{(L-1)} - \mathbf{y}_i\|^2.$$

More generally, use of non-constant weighting of the learning data in the cost functional incorporates *a priori* information into the learning problem. For example, a time-series prediction could be based on larger weighting of the more recent samples; especially if one tries to simulate a slowly varying dynamical process [60]. It is also straightforward to improve the LMS learning problem when more knowledge on the variance of the output-data is available [50, 79]. Finally, a locally weighted linear regression also adaptable for MLP learning was introduced in [68].

Difficulties with early stopping: One way to regularize the MLP is to use early stopping (cutting, termination) of the learning algorithm [8]. This may lead to a better (smoother) network with noisy learning data, especially when network is initialized with small random numbers in the almost linear region of a sigmoidal activation function. For example, early stopping may be due to a cross-validation technique when value of the cost functional in the test set starts increasing [75]. Moreover, the optimization process is usually stopped overhasty when a fixed number of epocs (iterations) with a fixed learning rate (line search parameter) is taken in the original backpropagation algorithm. Early stopping means certainly that the underlying optimization problem has not been solved precisely, and thus the results in Corollary 2 with their consequences (including the BODF result) are in general not valid. Formulation of general and robust conditions for stopping the optimization algorithm prematurely can be problematic, which usually leads to a very large number of tests for validating different networks [25]. It is also evident that using early stopping different solutions are obtained with different optimization methods. Finally, it was shown in [10] that if models with the same training error are chosen with equal probability, then the lowest generalization error is obtained by choosing the model corresponding to the training error minimum. This result is valid globally for linear models and locally, around the training error minimum, for nonlinear models.

4 Numerical results

Here, we describe numerical experiments based on the proposed techniques. All experiments are performed on an HP9000/J280 workstation (180 MHz PA8000 CPU) and the implementation is based on F77 (optimization and MLP realization) and MatLab [1] for data pre- and postprocessing.
As an optimization software we apply the limited memory quasi-Newton subroutine L-BFGS [87], which uses a sparse approximation of the BFGS-formula based inverse of the Hessian matrix and is intended for solving large nonlinear optimization problems efficiently. As a stopping criterion for the optimization we use

$$\frac{\mathcal{J}^k - \mathcal{J}^{k+1}}{\max\{|\mathcal{J}^k|, |\mathcal{J}^{k+1}|, 1\}} \leq 10^6 * epsmch,$$

where *epsmch* is the machine epsilon ($\sim 10^{-16}$ in our case). This choice reflects our intention to solve the optimization problem with (unnecessary) high precision. Moreover, according to the discussion in Section 3.4 we want to distinguish between the learning problem and the training algorithm so that application of a general optimization software as a black-box solver is justified and actually desirable.
There exists a large variety of different tests comparing backpropagation (gradient-decent with constant learning rate), conjugate gradient, and second-order methods (Gauss-Newton, Levenberg-Marquart, Hessian approximation, and quasi-Newton) for the MLP training (e.g., [8, 26, 51, 53, 72, 83]). The main ingredient in the L-BFGS-software is that due to the limited memory Hessian update, the storage requirement is only $\mathcal{O}(n)$, where n is the total amount of unknowns. For ordinary quasi-Newton methods, the $\mathcal{O}(n^2)$ memory consumption has been one of the main reasons to prevent the application of these methods for learning problems with larger network architectures.
Notice that a single gradient vector w.r.t. the unknown weights is obtained from the layer-wise gradient-matrices by simply reshaping and assembling.
In the numerical experiments, we only consider simple examples, because the emphasis here is on testing the algorithms and different formulations rather than on complex applications of NN's. Therefore, we also restrict ourselves to the perceptron with only one hidden layer although the actual implementation is suitable for any number of layers. As noted in [63] and references therein (see also [74]), there exist rather simple examples where MLP with two relatively small hidden layers can already reconstruct the given function whereas with one hidden layer an 'infinite' amount of hidden nodes are needed for a proper approximation. However, for training the network with several hidden layers one must solve significantly more complex optimization problem with an increased number of local minima. Hence, the dispute concerning the number of layers should consider the overall efficiency as a combination of approximation properties and training times.
In order to generate less regular nonlinear transformations by increasing the dimension n_1 of the hidden layer without affecting the scale of weights we choose $t_k(a)$, $k = 1, \ldots, n_1$, to be the activation functions for the hidden neurons [54]. From the physiological point of view, this is only a tiny change because real nervous systems are known to contain hundreds of different neuron types ([63] and references therein). Notice that even though our purpose here is to introduce some kind of ordering in the hidden layer by using different activation

functions for each neuron, the symmetry problem related to the hidden neurons (e.g., [8]) still remains as can be seen by a simple rescaling argument. Use of even more general mixture of different activation functions in the hidden layer is suggested in [86]. Finally, the possible danger when increasing the nonsmoothness of the hidden activation functions is that the convergence of second-order optimization methods is based on assuming C^2-continuity of the cost functional, which may for very large n_1 be almost violated.

4.1 Approximation of noisy function

The basic MLP

Example 1. First we consider the reconstruction of the function $f(x) = \sin(2\pi x)$, $x \in I = [0, 2\pi]$, which is corrupted with quasi-uniform, normally distributed random noise. The input-data points $\{x_i\}$ correspond to the uniform discretization of the interval I with the step size $h = 0.1$. Points in the output-data are taken as $y_i = f(x_i) + \delta\,\varepsilon_i$ for $\delta = 0.3$ and $\varepsilon_i \in \mathcal{N}(0,1)$. Altogether, we have in this example $N = 63$ and $n_2 = n_0 = 1$.

In the following experiments we have solved the optimization problem (14) with prescaled learning data into $[-1,1]$ starting from ten different (quasi-)random initial guesses from the same range $(-1,1)$ for the weight-matrices $(\mathbf{W}^1, \mathbf{W}^2)$. For an overview and study of different initialization techniques we refer to [76] and articles therein. Notice that according to [23] a quasi-Newton method can survey a larger amount of local minima than the BP- and CG-methods. Depending on application, this can be an advantage or disadvantage, but surely it indicates that either we have to start the training using multiple initial configurations or use some global optimization strategy as on outer iteration to enforce search through different minima.
The contents of the following tables include minimum 'min', maximum 'max', and average 'mean' values of the following quantities corresponding to the ten solutions of the optimization problem:

$\mathcal{J}^*$: Final value of the cost functional. If the minimum or maximum value (with tolerance $\varepsilon = 10^{-6}$) is scored more than once, the number of instances is included in parenthesis.

$|\bar{e}^*|$: Absolute value of the average output-error over the learning data as stated in Corollary 2, (i).

Its: Number of function/gradient evaluations during the optimization procedure.

CPU: CPU time in seconds for solving the optimization problem.

Finally, figures of the obtained network mappings corresponding to minimum and maximum values of $\mathcal{J}^*$ are given in Appendix B, where the mapping is illustrated using test set of data points $\{\tilde{x}_i\}$ generated on I using another step size $\tilde{h} = 0.13$.
Let us make some comments based on Table 1 and Figures 5–10:

Local minima: Even for the smallest network ($n_1 = 2$) and especially for larger ones there exist a lot of local minima in the optimization problem. Moreover, the arising local minima are strict in a sense that they correspond to truly different values of

	$\mathcal{J}^*$			$\lvert\bar{e}^*\rvert$			Its			CPU		
n_1	min	max	mean	min	max	mean	min	max	mean	min	max	mean
2	0.014 (2)	0.032	0.018	2e-8	4e-6	2e-6	41	234	126	0.23	0.52	0.39
3	0.013	0.024	0.014	4e-7	2e-5	4e-6	120	1079	427	0.42	0.80	0.60
4	0.012 (2)	0.014	0.013	1e-6	8e-6	4e-6	153	341	250	0.47	0.60	0.54
5	0.010	0.013	0.012	4e-7	2e-5	6e-6	237	804	430	0.55	0.76	0.64
6	0.010	0.013	0.011	3e-7	3e-5	8e-6	183	737	466	0.51	0.75	0.66
7	0.0086	0.013	0.010	2e-7	2e-5	7e-6	440	1541	927	0.67	0.95	0.80

Table 1: Computational results in Example 1 without regularization.

the cost functional and not just different representations (symmetries) of the same MLP-transformation.

Condition (i) **in Corollary 2:** Is valid with a precision related to the stopping criterion of the optimization.

Efficiency: Although the number of iterations in the optimization varies a lot, the CPU time remains always small in this small example.

Generalization: Best result according to Figures 5–10 is obtained using the MLP corresponding to the minimal value of $\mathcal{J}^*$ for $n_1 = 2$. However, MLP corresponding to the maximal value of $\mathcal{J}^*$ for $n_1 = 7$ gives also a reasonable result. This illustrates the difficulty of naming the 'optimal network' in a specific example. Moreover, from the large variation of the number of iterations we conclude that when a fixed number of iterations is taken in the learning algorithm, one has no knowledge on the error between the obtained weight-matrices and the true (local) solution of the optimization problem (cf. the discussion in Section 3.4).

Regularization of MLP using WD

A problem is said to be well-posed in the sense of Hadamard if i) it has a solution, ii) the solution is unique, iii) the solution is stable w.r.t. data. If a given problem is not well-posed a usual way to improve its structure is regularization (in general, see [78, 69]; in connection with neural networks, see [24, 25, 29, 59]), which usually refers to increasing smoothness of a solution.

The existence of local minima shows that the LMS cost functional to be minimized in the learning problem is mathematically speaking highly non-convex, containing a large amount of local minima. A natural remedy for this problem is to increase coercivity of the cost functional w.r.t. the unknowns. In the simplest form this can be accomplished by introducing an extra penalization term in the cost functional, and usually in connection with neural networks this yields some kind of a weight decay (WD) technique [25]. As a consequence, this also imposes some restriction on the generality (universality) of the MLP-transform to prevent overlearning yielding to a special kind of pruning [5, 33, 36, 43, 46, 61, 62, 70, 73, 79, 85].

Hence, we enhance the optimization problem for network training with a WD-term $\mathcal{R}(\mathbf{W}^l_{ij})$. More precisely, we use the simplest possible strictly convex form w.r.t the unknown weights

by considering initially $\mathcal{R}(\mathbf{W}_{ij}^l) = \beta/2 \sum_{l,i,j} (\mathbf{W}_{ij}^l)^2$, where β is a weight decay parameter representing our confidence in the data. Here, the choice of having only a single coefficient makes sense, because the network inputs and the outputs of the hidden layer are in the same scale due to the applied prescaling. We remind that in general the 'best' value of β is related to both the complexity of the MLP-transformation and the (usually unknown) amount of noise contained in the learning data [69]. However, one can, e.g., apply different cross-validation techniques for obtaining an effective choice of β (for a general discussion, see [8, 64]; precisely in our setting, see [66]).
In addition to strict convexity some particular reasons for choosing the proposed form of WD with the quadratic penalty function $p(w) = |w|^2$ are:

- Because single weights are treated separately, it is trivial to include the corresponding derivative-matrices in the optimality conditions.
- The proposed form forces weights in the neighborhood of zero (similarly to prior distributions with zero mean suggested in [56]) thus balancing their scale in a gradient-based optimization algorithm [11]. The smoothing property is due to the fact that the activation functions $s_k(a)$ and $t_k(a)$ are nearly linear around zero, although size of the linear region is decreasing as k is increasing (cf. Figure 1). Furthermore, the first derivatives of both sigmoidal activation functions are most informative around zero. Thus, the proposed form of WD is also helpful to prevent saturation of weights by enforcing them in the neighborhood of this transient region [45, 80].
- In [66] the conclusion from the numerical experiments was that the combination of quasi-Newton optimization algorithm and quadratic WD drastically improved both the convergence of the training algorithm and the generalization performance of the trained MLP.

One drawback of quadratic WD is that it produces weight matrices with many small components even if a choice of one large weight instead of a group of small ones could be sufficient. This can yield unnecessary large networks, even if the overlearning can be prevented. On the other hand, this property increases the fault tolerance of the trained network due to the so-called *graceful degradation* [63].
Let us comment some of the difficulties resulting from a few other forms of WD for individual weights suggested in the literature [24, 25, 66]:

l^1**−formulation:** Each weight w is regularized using penalization $p(w) = |w|$. This function is convex, but not strictly convex. A severe difficulty is that because the derivative of $p(w)$ is multivalued for $w = 0$, the resulting optimization problem is nonsmooth, i.e., only subdifferentiable [52]. In general, function $|w|^p$ for $1 < p < 2$ only belongs to Hölder space $C^{1,p-1}$ [22], so that assumptions for convergence of gradient-descent (on batch-mode Lipschitz continuity of gradient [58], for on-line stochastic iteration C^2 continuity [28]), CG (Lipschitz continuity of gradient [58]), and especially quasi-Newton methods (C^2-continuity [58]) are violated. As documented, e.g., in [66] (MLP) and [39] (image restoration) this yields nonconvergence of training algorithm when cost functional does not fulfil the required smoothness assumption. Furthermore, even if smoothed counterpart $\sqrt{w^2 + \varepsilon}$ for $\varepsilon > 0$ is introduced, this formulation is either (for small ε) too close to the nonsmooth case so that again the convergence of ordinary solution methods fail, or, otherwise (for larger ε), the smoothed formulation differs substantially from the original one. For such nonsmooth optimization problems one needs to develop special algorithms [37, 38, 39]. Finally, these same difficulties also concern the so-called robust backpropagation where the error function is defined as $\sum_i \|\mathcal{N}(\mathbf{x}_i) - \mathbf{y}_i\|_{l^1}$ (e.g., [44] and articles therein).

Mixed WD: Each weight is regularized using mixed penalization $p(w) = w^2/(1+w^2)$. This form is, firstly, non-convex producing even more local minima to the optimization problem and, secondly, it favors both small and large weights thus corrupting the gradient by destroying the balance in scales of different weights.

However, Corollary 2 and the description of the MLP-transformation (role of bias-terms as a shift from origin) raise the question "Which components of weight-matrices $\{\mathbf{W}^l\}_{l=1}^{L}$ should be regularized and which not?" Certainly, if condition (i) in Corollary 2 is required for the resulting MLP, one should not include the bias-terms of the final layer $\mathbf{W}^L$ in WD. Similarly, for conditions (i)–(ii) both to hold one should exclude all components of $\mathbf{W}^L$ from WD. These questions will be studied in the following experiments. According to the above discussion, we apply the proposed form of WD only for selected components of the weight-matrices $(\mathbf{W}^1, \mathbf{W}^2)$. We distinguish between four cases:

I Regularize all other components except the bias-terms $\mathbf{W}_0^2$ in the weight-matrix $\mathbf{W}^2$.

II Exclude all components of $\mathbf{W}^2$ from the regularization.

III Exclude all bias-terms of $(\mathbf{W}^1, \mathbf{W}^2)$ from the regularization.

IV Exclude all components of $\mathbf{W}^2$ and bias-terms of $\mathbf{W}^1$ from the regularization.

Notice that WD according to case III is used in [30] for an MLP-classifier.

Remark 4. Let us state one further observation concerning the non-regularization of $\mathbf{W}_0^2$. Using the error-average formula $\frac{1}{N}\sum_{i=1}^{N} \mathbf{e}_i^* = \mathbf{O}$ of Corollary 2 and the expression for $\mathbf{e}_i^*$ according to (18) yields (cf. [8])

$$\mathbf{W}_0^{2,*} = -\frac{1}{N}\sum_{i=1}^{N}[\mathbf{W}_1^{2,*}\,\mathcal{F}(\mathbf{W}^{1,*}\,\hat{\mathbf{x}}_i) - \mathbf{y}_i].$$

	$\mathcal{J}_\beta^*$			$\lvert\bar{e}^*\rvert$			Its			CPU		
n_1	min	max	mean	min	max	mean	min	max	mean	min	max	mean
2	0.022 (5)	0.025 (5)	0.023	3e-8	4e-6	1e-6	48	84	65	0.25	0.35	0.30
3	0.0195	0.0199 (2)	0.0197	3e-8	5e-6	2e-6	76	149	101	0.33	0.43	0.37
4	0.017	0.019	0.018	1e-7	4e-6	2e-6	88	208	146	0.38	0.51	0.44
5	0.017	0.018	0.017	1e-7	2e-5	7e-6	142	245	188	0.46	0.55	0.50
6	0.016	0.017 (2)	0.017	5e-7	1e-5	6e-6	145	382	244	0.47	0.63	0.55
7	0.016	0.017	0.016	2e-7	2e-5	5e-6	227	395	327	0.57	0.65	0.62

Table 2: Computational results in Example 1 for regularization **I**.

n_1	$\mathcal{J}_\beta^*$ min	$\mathcal{J}_\beta^*$ max	$\mathcal{J}_\beta^*$ mean	$\lvert\bar{e}^*\rvert$ min	$\lvert\bar{e}^*\rvert$ max	$\lvert\bar{e}^*\rvert$ mean	Its min	Its max	Its mean	CPU min	CPU max	CPU mean
2	0.015	0.016	0.015	2e-6	8e-5	2e-5	67	554	231	0.33	0.67	0.49
3	0.014	0.015	0.015	2e-7	1e-5	3e-6	131	1009	383	0.44	0.78	0.59
4	0.014	0.015	0.015	8e-8	2e-5	8e-6	257	963	414	0.48	0.79	0.61
5	0.014	0.015 (2)	0.014	2e-6	3e-5	1e-5	329	1283	832	0.61	0.87	0.75
6	0.013	0.014	0.014	3e-6	3e-5	1e-5	607	2402	1331	0.72	1.10	0.88
7	0.013	0.014	0.014	1e-7	1e-5	4e-6	1213	2472	1599	0.86	1.13	0.96

Table 3: Computational results in Example 1 for regularization **II**.

Hence, increased coercivity and thereby uniqueness w.r.t to $(\mathbf{W}_1^2, \mathbf{W}^1)$ immediately affects the uniqueness of the bias $\mathbf{W}_0^2$ as well.

The results corresponding to the above cases are presented in Tables 2–5 (where $\mathcal{J}_\beta^*$ refers to the value of the cost functional containing the regularization term) and in Figures 11–34. For all experiments we have chosen $\beta = 10^{-3}$ according to some prior tests, which also indicated that the obtained results in these examples were not very sensitive to the choice of β.
Let us give some observations based on Tables 2–5 and Figures 11–36 in Appendix B:

Local minima: Even if the regularization improves the convexity of the cost functional there still exist a lot of local minima in the optimization problem.

Condition (i) **in Corollary 2:** Is valid with a precision related to the stopping criterion of the optimization.

Efficiency: By means of number of iterations and the CPU time regularization methods **I** and **III** are better and more robust than the regularization methods **II** and **IV**.

Generalization: All regularization methods improve the obtained results compared to the original formulation by preventing oscillation in the final mapping generated by the MLP. When n_1 is increased, regularization methods **I** and **III** seem to be more stable than regularization methods **II** and **IV**. Moreover, for small n_1 the minimal cost function values are scored more than once for methods **I** and **III** suggesting an improved convexity compared to methods **II** and **IV**.

n_1	$\mathcal{J}_\beta^*$ min	$\mathcal{J}_\beta^*$ max	$\mathcal{J}_\beta^*$ mean	$\lvert\bar{e}^*\rvert$ min	$\lvert\bar{e}^*\rvert$ max	$\lvert\bar{e}^*\rvert$ mean	Its min	Its max	Its mean	CPU min	CPU max	CPU mean
2	0.022 (5)	0.025 (5)	0.023	1e-7	3e-6	1e-6	49	93	71	0.25	0.36	0.30
3	0.019 (4)	0.025	0.019	3e-8	7e-6	2e-6	73	173	109	0.33	0.48	0.39
4	0.017 (2)	0.019	0.018	3e-7	6e-6	2e-6	87	172	121	0.36	0.48	0.41
5	0.017	0.019	0.017	3e-7	1e-5	4e-6	80	205	141	0.36	0.51	0.45
6	0.016	0.017	0.017	3e-7	7e-6	3e-6	104	199	153	0.40	0.51	0.47
7	0.016	0.018	0.016	2e-6	3e-5	9e-6	118	537	269	0.44	0.70	0.56

Table 4: Computational results in Example 1 for regularization **III**.

	$\mathcal{J}_\beta^*$			$\lvert\bar{e}^*\rvert$			Its			CPU		
n_1	min	max	mean	min	max	mean	min	max	mean	min	max	mean
2	0.015	0.016	0.015	4e-7	2e-5	1e-5	73	913	235	0.31	0.75	0.47
3	0.014	0.016	0.015	9e-7	5e-5	1e-5	110	953	352	0.39	0.77	0.56
4	0.014	0.015	0.015	2e-7	3e-5	9e-6	151	744	344	0.47	0.74	0.58
5	0.014	0.015 (2)	0.014	9e-8	1e-5	5e-6	244	2436	1176	0.55	1.09	0.82
6	0.013	0.014	0.014	2e-7	2e-5	7e-6	361	2135	1184	0.62	1.05	0.84
7	0.012	0.014	0.014	1e-8	2e-5	7e-6	321	3119	1368	0.61	1.19	0.89

Table 5: Computational results in Example 1 for regularization **IV**.

Conclusion: From the tested regularization methods **I** and **III** seem to be more preferable than **II** and **IV**. When comparing **I** and **III** we remind that the prescaling of the data into the range $[-1, 1]$ already decreases the significance of the bias-term $\mathbf{W}_0^1$.

4.2 Classification

Example 2. As a second example, we consider the well-known test case to classify Iris flowers (cf. [25]) according to measurements obtained from the UCI repository [9]. In this example, we have $n_2 = 3$ (number of classes), $n_0 = 4$ (number of features) and initially 50 samples from each of the three classes. We remind here that due to the choice of the 'k-tanh' activation functions prescaling of the output-data $\{y_i\}$ into the range $[-1, 1]$ destroys the linear independence between different classes.

In Tables 6–11, we have solved the classification (optimization) problem again ten times using 40 random samples from each class as the learning set and the remaining 10 samples from each class as the test set. These two data sets have been formed using five different random permutations of the initial data realizing a simple resampling procedure (that could give rise to an actual cross-validation technique). Hence, we have $N = 120$ and the remaining 30 samples in the test set. In Tables 6–11 the first column 'P' contains the permutation index. In the second column 'C_L', minimum, maximum and (rounded) mean values for the number of false classifications in the learning set are given. The third column 'C' includes numbers of false classifications in the learning and test sets (denoted with C_L and C_T) for the 'optimal' perceptron $\mathcal{N}_P^*$ satisfying $C_L(\mathcal{N}_P^*) + C_T(\mathcal{N}_P^*) \leq C_L(\mathcal{N}_P) + C_T(\mathcal{N}_P)$ over all networks encountered during the ten optimization runs for the current permutation P. The overall quality of $\mathcal{N}_P^*$ for each permutation is measured using the total sum of misclassifications $C_L(\mathcal{N}^*) + C_T(\mathcal{N}^*)$.
In Tables 9–11 we have added noise to the inputs of the learning data. More precisely, we have first computed class means w.r.t. each of the four features within the three classes. After that the vector of class means has been multiplied componentwise with a noise vector $\delta\,\varepsilon_i$ for $\delta = 0.3$ and $\varepsilon_i \in \mathcal{N}(0, 1)$ and this has been added to (unscaled) learning data $\{\mathbf{x}_i\}_{i\in L}$ obtained after permutation of the initial data.
To this end, we derive some final observations based on the computational results, especially in Example 2:

Local minima: In all numerical tests for unregularized and regularized learning problems with methods I and III the L-BFGS optimization algorithm was convergent. This strongly suggests that using the proposed setting of prescaling and MLP-architecture we were able to deal with the flat error surfaces during the training. Furthermore, because usually in all ten testruns of the learning problem a different value of the cost functional was obtained, the different results corresponded to 'true' local minima instead of just the symmetrical presentations of the same MLP-transformation.

Efficiency: By comparing the number of iterations and the CPU time the unregularized problem seems to be easier to solve than the regularized one (with given β) for the clean learning data. On average, there is no real difference between the unregularized and regularized problems for the noisy learning data, but the variation in the number of iterations is significantly larger for the unregularized formulation.

Comparison: There seems to be no significant difference between the quality of perceptrons resulting from the unregularized and regularized problems. This suggests that the Iris learning data contains only a few cases with non-Gaussian degradations. Moreover, according to the observations in Example 1 one reason for the quite similar behaviour can be the size of n_1 which is not that large compared to n_0 and n_2. Finally, we cannot favor either of the two regularization methods I and III according to these results. Let us further mention that especially when using the regularization method II (and in some cases IV) the L-BFGS-software sometimes failed in optimization indicating that the optimization problem for these formulations may have a higher complexity due to increased nonconvexity and flatness problems.

Generalization: The amount of false classifications in the test set is between 0–3 , i.e. between 0%–10% in all test runs. A single preferable choice (for different strategies to combine multiple networks as an ensemble, see, e.g., [27, 49]) for an classifier would probably be the one with median of false classifications in C_T obtained using permutation one or four, even if with the fifth permutation an optimal one was found according to the given data. Notice that the amount of variation in C_T over the different permutations provides some information on the quality of the data.

	C_L			C		Its			CPU		
P	min	max	mean	C_L	C_T	min	max	mean	min	max	mean
1	0	0	0	0	2	109	534	239	0.37	0.60	0.47
2	0	0	0	0	3	81	664	290	0.30	0.65	0.49
3	0	1	1	0	1	106	1243	492	0.35	0.93	0.57
4	0	0	0	0	2	129	891	349	0.36	0.75	0.51
5	0	1	1	0	0	92	702	286	0.31	0.67	0.48
mean	0	0.4	0.4	0	1.6	103	807	331	0.34	0.72	0.50

Table 6: Computational results in Example 2 without regularization for $n_1 = 9$.

P	C_L min	C_L max	C_L mean	C C_L	C C_T	Its min	Its max	Its mean	CPU min	CPU max	CPU mean
1	0	0	0	0	2	517	1280	843	0.60	0.93	0.76
2	0	0	0	0	3	443	814	551	0.57	0.72	0.63
3	0	0	0	0	0	742	1797	1047	0.68	1.04	0.82
4	0	0	0	0	2	426	1451	792	0.56	0.91	0.71
5	0	1	0	0	0	646	1604	884	0.69	1.00	0.76
mean	0	0.2	0	0	1.4	555	1389	823	0.62	0.92	0.74

Table 7: Computational results in Example 2 for regularization **I** with $\varepsilon = 10^{-3}$ and $n_1 = 9$.

P	C_L min	C_L max	C_L mean	C C_L	C C_T	Its min	Its max	Its mean	CPU min	CPU max	CPU mean
1	0	0	0	0	2	369	939	543	0.56	0.77	0.61
2	0	0	0	0	3	484	923	602	0.58	0.80	0.64
3	0	2	1	0	0	418	868	677	0.56	0.78	0.67
4	0	0	0	0	3	382	643	507	0.54	0.69	0.60
5	0	3	1	0	0	413	973	619	0.56	0.82	0.64
mean	0	1	0.4	0	1.6	413	869	590	0.56	0.77	0.63

Table 8: Computational results in Example 2 for regularization **III** with $\varepsilon = 10^{-3}$ and $n_1 = 9$.

5 Conclusions

5.1 Summary of the present paper

We have studied the transformation realized by the MLP-network. A compact, layer-wise description for the action of MLP has been given. Based on this formalism, optimality conditions for the LMS cost functional have been presented and their consequences derived and discussed. In the numerical experiments, study of different regularization methods based on the earlier findings has been performed. Our numerical approach used can be summarized as:

- For balancing the sensitivities w.r.t all weights, prescale the input-output -data into the equal range of activation functions.

P	C_L min	C_L max	C_L mean	C C_L	C C_T	Its min	Its max	Its mean	CPU min	CPU max	CPU mean
1	0	1	0	0	2	275	3008	968	0.49	1.19	0.73
2	0	1	1	0	2	240	460	339	0.47	0.57	0.52
3	0	3	1	0	1	175	2680	948	0.41	1.16	0.73
4	0	1	1	0	2	146	1655	589	0.39	0.98	0.61
5	0	2	1	0	0	284	5464	927	0.50	1.42	0.65
mean	0	1.6	0.8	0	1.4	224	2653	754	0.45	1.06	0.65

Table 9: Computational results in Example 2 without regularization for $n_1 = 9$ and noisy learning set.

P	C_L min	C_L max	C_L mean	C C_L	C C_T	Its min	Its max	Its mean	CPU min	CPU max	CPU mean
1	0	1	0	0	2	492	1196	724	0.59	0.88	0.68
2	0	0	0	0	3	486	1059	782	0.58	0.81	0.70
3	0	0	0	0	1	579	982	739	0.62	0.78	0.69
4	0	1	0	0	2	616	1120	862	0.64	0.84	0.74
5	0	1	1	0	0	464	1211	756	0.58	0.88	0.69
mean	0	0.6	0.2	0	1.6	527	1114	773	0.60	0.84	0.70

Table 10: Computational results in Example 2 for regularization **I** with $\varepsilon = 10^{-3}$, $n_1 = 9$ and noisy learning set.

P	C_L min	C_L max	C_L mean	C C_L	C C_T	Its min	Its max	Its mean	CPU min	CPU max	CPU mean
1	0	1	0	0	2	434	1024	710	0.56	0.80	0.68
2	0	2	1	0	3	365	941	580	0.54	0.76	0.62
3	0	3	1	0	1	464	800	653	0.58	0.71	0.65
4	0	2	1	0	2	386	1007	725	0.54	0.83	0.69
5	0	1	1	0	0	458	1003	690	0.58	0.79	0.67
mean	0	1.8	0.8	0	1.6	421	955	672	0.56	0.78	0.66

Table 11: Computational results in Example 2 for regularization **III** with $\varepsilon = 10^{-3}$, $n_1 = 9$ and noisy learning set.

- To prevent overlearning use weight decay, but in such a form that every trained mapping has zero error on average according to Corollary 2. This also requires that the optimization problem must be solved with a fixed precision.
- To deal with multiple local minima, solve the learning problem starting from several initial configurations and determine a measure for classifying these results.
- To deal with inconsistent data, use some kind of outer iterative procedure, which is here realized using resampling with permutation. Again one must construct a way (appropriate measures) to pick up the best one or combine the different ones.

Finally, we think that the given implementation of MLP together with the L-BFGS software as an optimizer is well-suited for an efficient local resolution of the learning problem also for significantly larger networks. One of the main challenges for future research is to study enhanced formulations containing less local minima for the MLP learning problem.

5.2 Discussion

To conclude and generalize this prensentation, let us consider a rough scheme of artificial intelligence based on the data analysis as sketched in Figure 4. In the scheme we have the following characteristics:

Application: characterized by available data

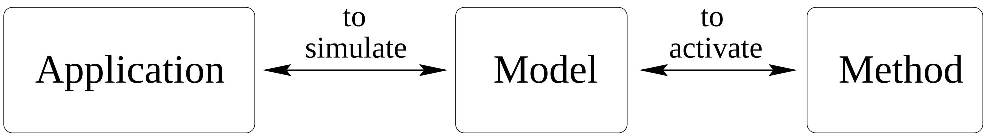

Figure 4: Scheme of data analysis.

Model: characterized by free parameters; consists of

Architecture: complexity of distribution

Learning problem: quality of data

Method: characterized by training algorithms

In MLP, the given input-output pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$ contain the available data. Free parameters correspond to unknown weights in the weight matrices and architecture is represented by amount and dimension of the hidden layers with the chosen activation functions. Learning problem in the present work refers to LMS optimization problem and training algorithm should solve this problem in order to adjust the network weights.
It is necessary to follow the above steps in the forward mode, i.e., when seeking and developing models and their training algorithms to simulate the phenomena behind the application. However, success of the realization (backward or reverse mode within the schema) is often analysed by qualifying the system solely by its performance in the application thus suppressing the relation between the model and the method. As a typical example, we have again the case when the optimization problem for the MLP-training is solved with an unknown precision by taking only fixed number of epocs, and the resulting network is validated, e.g., by measuring the amount of false classifications in the test set. Even if this can work well in practice (especially with cross-validation), ideal goal in the reverse mode is to derive a model that i) agrees with the complexity of the approximated distribution and ii) takes into account the quality of data in the learning problem formulation. If this is accomplished, then the only purpose of a method is to estimate the free parameters with a suitable, fixed precision.
In neural networks literature it is sometimes mentioned that *by improving the training algorithm one can improve the performance of MLP* (usually generalization ability). However, if such a phrase is referring for speeding up the training algorithm for solving the optimization problem (using better choice of learning rate parameters, cf. [15]), then the formulation of the learning problem remains unchanged. Hence, actually one is either changing the strategy behind early stopping or just converging to another local minimum than before. Only if the learning problem is truly altered, e.g., by changing the error function to be minimized, then one is really considering another model according to Figure 4.
This leads us to a particular problem with the MLP-networks, which is the role of the term 'optimality'. Namely, the MLP satisfying the optimality conditions of the LMS learning problem may not be the best generalizer or classifier in the considered application, vice versa, by means of optimization the non-optimal solutions may produce the 'best' networks in terms of the application (cf. results in Section 4). Furthermore, optimality of a training

algorithm should refer to the speed of convergence rather than to the final result which is determined by the model.

The existence of local minima and remedies for them is an extensively studied subject in the MLP literature (e.g., [3, 7, 8, 14, 20, 41, 53, 72] and articles therein). However, Corollary 2 and the previously mentioned BODF-result in [65] show that in any case every local minimum contains a fair amount of 'optimality' w.r.t. the given data. The phrase 'false local minima' refers to bad generalization that may be observed for some solutions of the learning problem. The difficulty is that in many cases one has not a precise knowledge on the relation between the error function and false local minima. More precisely, the smallest value of the error function, the global minimum, may in the sense of generalization properties be just another false local minimum. Typically this happens when the architecture is too flexible so it can overlearn the data by capturing the erroneous, noisy behaviour of the system (cf. Example 1 in Section 4 without regularization). In this case, improvement of the minimization process, e.g., by using methods of global optimization (genetic algorithms, simulated annealing, etc.) does not necessarily produce better model. It can have even the opposite effect. Moreover, the earlier mentioned model selection result in [10] nominating the training error minimizer as the best model for generalization over models which are chosen with equal probability was only valid locally for nonlinear models and, thus, it cannot be applied to compare different local solutions. The conflict between the learning problem minimization and best generalization really calls for new improved models where these two aims could be in agreement.

Altogether, one should try to develop such formulations for the network learning which at the same time allow the treatment of the learning problem as a deterministic optimization problem due to the obtainable computing efficiency and also contain appropriate measures for the quality of the learning data and the obtained model in the considered, specific example. One should keep in mind that usage of one model (like MLP with a predescribed architecture) is based on the assumption that the joint distribution $p(x, y) = p(y|x)\, p(x)$ remains the same, i.e., both the input distribution $p(x)$ and the conditional distribution $p(y|x)$ are fixed throughout the samples. For example, the Bayesian framework for MLP as firstly introduced by Mackay [50] (for a review, see [77] and articles therein; see also [34, 42, 73]) and further developed by Neal [56], other statistical methods [2, 4] and even more general approaches [71] are steps towards an improved model selection in the expense of increased computational complexity. For statistical approaches one problem is whether the prior assumptions concerning the chosen form of density functions are valid and can be rigorously tested on the given data. If this is not the case (especially, for mixture models consisting of different basic density-distributions), then a hybrid combination of cross-validation and resampling producing different views on the data is a reasonable choice. By slightly modifying the central questions in software engineering, we should always ask ourselves "Are we training the right model?", "Are we training the model right?"

References

[1] *Getting Started with MATLAB*, MathWorks, Inc., Natick, MA, 1997.

[2] E. Alpaydm, *Combined* 5×2 *cv F test for comparing supervised classification learning algorithms*, Neural Computation, 11 (1999), pp. 1885–1892.

[3] N. Ampazis, S. J. Perantonis, and J. G. Taylor, *Dynamics of multilayer networks in the vicinity of temporary minima*, Neural Networks, 12 (1999), pp. 43–58.

[4] U. Anders and O. Korn, *Model selection in neural networks*, Neural Networks, 12 (1999), pp. 309–323.

[5] B. S. Arad and A. El-Amawy, *On fault tolerant training of feedforward neural networks*, Neural Networks, 10 (1997), pp. 539–553.

[6] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, New York, 1982.

[7] M. Bianchini, M. Gori, and M. Maggini, *On the problem of local minima in recurrent neural networks*, IEEE Trans. Neural Networks, 5 (1994), pp. 167–177.

[8] C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

[9] C. L. Blake and C. J. Merz, *UCI repository of machine learning databases*, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[10] Z. Cataltepe, Y. S. Abu-Mostafa, and M. Magdon-Ismail, *No free lunch for early stopping*, Neural Computation, 11 (1999), pp. 995–1009.

[11] S. Cavalieri and O. Mirabella, *A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks*, Neural Networks, 12 (1999), pp. 91–106.

[12] C. L. P. Chen, *A rapid supervised learning neural network for function interpolation and approximation*, IEEE Trans. Neural Networks, 7 (1996), pp. 1220–1230.

[13] D. S. Chen and R. C. Jain, *A robust back propagation learning algorithm for function approximation*, IEEE Trans. Neural Networks, 5 (1994), pp. 467–479.

[14] F. M. Coetzee and V. L. Stonick, *Contrast enhancement for backpropagation*, IEEE Trans. Neural Networks, 7 (1996), pp. 318–325.

[15] H. Dai and C. MacBeth, *Effects of learning parameters on learning procedure and performance of a BPNN*, Neural Networks, 10 (1997), pp. 1505–1521.

[16] J. de Villiers and E. Barnard, *Backpropagation neural networks with one and two hidden layers*, IEEE Trans. Neural Networks, 1 (1992), pp. 136–141.

[17] P. De Wilde, *The magnitude of the diagonal elements in neural networks*, Neural Networks, 10 (1997), pp. 499–504.

[18] M. Di Martino, S. Fanelli, and M. Protasi, *Exploring and comparing the best "direct methods" for the efficient training of MLP-networks*, IEEE Trans. Neural Networks, 7 (1996), pp. 1497–1502.

[19] K. I. Diamantaras and S. Y. Kung, *Principal Component Neural Networks: Theory and Applications*, John Wiley & Sons, New York, 1996.

[20] Y. Fukuoka, H. Matsuki, H. Minamitani, and A. Ishida, *A modified back-propagation method to avoid false local minima*, Neural Networks, 11 (1998), pp. 1059–1072.

[21] A. A. Ghorbani and V. C. Bhavsar, *Incremental communication for multilayer neural networks*, IEEE Trans. Neural Networks, 6 (1995), pp. 1375–1385.

[22] D. Gilbarg and N. S. Trudinger, *Elliptic Partial Differential Equations of Second Order*, Grundlehren der mathematischen Wissenschaften 224, Springer-Verlag, Berlin Heidelberg, 1983.

[23] D. Gorse, A. J. Shepherd, and J. G. Taylor, *The new ERA in supervised learning*, Neural Networks, 10 (1997), pp. 343–352.

[24] C. Goutte and L. K. Hansen, *Regularization with a pruning prior*, Neural Networks, 10 (1997), pp. 1053–1059.

[25] A. Gupta and S. M. Lam, *Weight decay backpropagation for noisy data*, Neural Networks, 11 (1998), pp. 1527–1137.

[26] M. T. Hagan and M. B. Menhaj, *Training feedforward networks with the Marquardt algorithm*, IEEE Trans. Neural Networks, 5 (1994), pp. 989–993.

[27] S. Hashem, *Optimal linear combinations of neural networks*, Neural Networks, 10 (1997), pp. 599–614.

[28] S. Haykin, *Neural Networks; A Comprehensive Foundation*, Macmillan College Publishing Company, New York, 1994.

[29] M. Hintz-Madsen, L. K. Hansen, J. Larsen, M. W. Pedersen, and M. Larsen, *Neural classifier construction using regularization, pruning and test error estimation*, Neural Networks, 11 (1998), pp. 1659–1670.

[30] L. Holmström, P. Koistinen, J. Laaksonen, and E. Oja, *Neural and statistical classifiers - taxonomy and two case studies*, IEEE Trans. Neural Networks, 8 (1997), pp. 5–17.

[31] P. J. Huber, *Robust Statistics*, Wiley, New York, 1981.

[32] D. Husmeier and J. G. Taylor, *Predicting conditional probability densities of stationary stochastic time series*, Neural Networks, 10 (1997), pp. 479–497.

[33] J.-N. Hwang, S.-S. You, S.-R. Lay, and I.-C. Jou, *The cascade-correlation learning: A projection pursuit learning perspective*, IEEE Trans. Neural Networks, 7 (1996), pp. 278–289.

[34] R. A. Jacobs, F. Peng, and M. A. Tanner, *A Bayesian approach to model selection in hierarchical mixtures-of-experts architectures*, Neural Networks, 10 (1997), pp. 231–241.

[35] N. Japkowicz, S. J. Hanson, and M. A. Gluck, *Nonlinear autoassociation is not equivalent to PCA*, Neural Computation, 12 (2000), pp. 531–545.

[36] P. P. Kanjilal and D. N. Banerjee, *On the application of orthogonal transformation for the design and analysis of feedforward networks*, IEEE Trans. Neural Networks, 6 (1995), pp. 1061–1070.

[37] T. Kärkkäinen and K. Majava, *Nonmonotone and monotone active-set methods for image restoration, Part 1: Convergence analysis*, J. Optimiz. Theory and Appl., 106 (2000), pp. 61–80.

[38] ——, *Nonmonotone and monotone active-set methods for image restoration, part 2: Numerical results*, J. Optimiz. Theory and Appl., 106 (2000), pp. 81–105.

[39] T. Kärkkäinen, K. Majava, and M. M. Mäkelä, *Comparison of formulations and solution methods for image restoration problems*, Report No. B 14, University of Jyväskylä, Department of Mathematical Information Technology, 2000.

[40] T. Kärkkäinen and J. Toivanen, *Building blocks for odd-even multigrid with applications to reduced systems*, J. Comp. Appl. Math., (2000). to appear.

[41] D. A. Karras and S. J. Perantonis, *An efficient constrained training algorithm for feedforward networks*, IEEE Trans. Neural Networks, 6 (1995), pp. 1420–1434.

[42] A. Kehagias and V. Petridis, *Predictive modular neural networks for time series classification*, Neural Networks, 10 (1997), pp. 31–49.

[43] R. A. Koene and Y. Takane, *Discriminant component pruning: Regularization and interpretation of multilayered backpropagation networks*, Neural Computation, 11 (1999), pp. 783–802.

[44] B. Kosko, *Neural Networks and Fuzzy Systems; A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, Englewood Cliffs, N.J., 1992.

[45] T. M. Kwon and H. Cheng, *Contrast enhancement for backpropagation*, IEEE Trans. Neural Networks, 7 (1996), pp. 515–524.

[46] C. W. Lee, *Training feedforward neural networks: An algorithm giving improved generalization*, Neural Networks, 10 (1997), pp. 61–68.

[47] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*, Neural Networks, 6 (1993), pp. 861–867.

[48] K. Liano, *Robust error measure for supervised neural network learning with outliers*, IEEE Trans. Neural Networks, 7 (1996), pp. 246–250.

[49] Y. Liu and X. Yao, *Ensemble learning via negative correlation*, Neural Networks, 12 (1999), pp. 1399–1404.

[50] D. J. C. MacKay, *A practical Bayesian framework for backpropagation networks*, Neural Computation, 4 (1992), pp. 448–472.

[51] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis, *Improving the convergence of the backpropagation algorithm using learning rate adaptation methods*, Neural Computation, 11 (1999), pp. 1769–1796.

[52] M. M. Mäkelä and P. Neittaanmäki, *Nonsmooth Optimization; Analysis and Algorithms with Applications to Optimal Control*, World Scientific, Singapore, 1992.

[53] J. J. McKeown, F. Stella, and G. Hall, *Some numerical aspects of the training problem for feed-forward neural nets*, Neural Networks, 10 (1997), pp. 1455–1463.

[54] D. McLean, Z. Bandar, and J. D. O'Shea, *An empirical comparison of back propagation and the RDSE algorithm on continuously valued real world data*, Neural Networks, 11 (1998), pp. 1685–1694.

[55] J. O. Moody and P. J. Antsaklis, *The dependence identification neural network construction algorithm*, IEEE Trans. Neural Networks, 7 (1996), pp. 3–15.

[56] R. M. Neal, *Bayesian Learning for Neural Networks*, Springer, New York, 1996.

[57] V. Nedeljković, *A novel multilayer neural networks training algorithm that minimizes the probability of classification error*, IEEE Trans. Neural Networks, 4 (1993), pp. 650–659.

[58] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, 1999.

[59] D. Ormoneit, *A regularization approach to continuous learning with an application to financial derivatives pricing*, Neural Networks, 12 (1999), pp. 1405–1412.

[60] D. C. Park, M. A. El-Sharkawi, and R. J. Marks, II, *An adaptively trained neural network*, IEEE Trans. Neural Networks, 2 (1991), pp. 334–345.

[61] G. E. Peterson, D. C. St. Clair, S. R. Aylward, and W. E. Bond, *Using Taguchi's method of experimental design to control errors in layered perceptrons*, IEEE Trans. Neural Networks, 6 (1995), pp. 949–961.

[62] R. Reed, *Pruning algorithms - a survey*, IEEE Trans. Neural Networks, 4 (1993), pp. 740–747.

[63] R. D. Reed and R. J. Marks, II, *Neural Smithing; Supervised Learning in Feedforward Artificial Neural Networks*, The MIT Press, Cambridge, 1999.

[64] R. Rohas, *Neural Networks; A Systematic Introduction*, Springer-Verlag, Berlin Heidelberg, 1996.

[65] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter, *The multilayer perceptron as an approximation to a Bayes optimal discriminant function*, IEEE Trans. Neural Networks, 1 (1990), pp. 296–298.

[66] K. Saito and R. Nakano, *Second-order learning algorithm with squared penalty term*, Neural Computation, 12 (2000), pp. 709–729.

[67] F. Scarselli and A. C. Tsoi, *Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results*, Neural Networks, 11 (1998), pp. 15–37.

[68] S. Schaal and C. G. Atkeson, *Constructive incremental learning from only local information*, Neural Computation, 10 (1998), pp. 2047–2084.

[69] O. Scherzer, H. W. Engl, and K. Kunisch, *Optimal a posteriori parameter choice for Tikhonov regularization for solving nonlinear ill-posed problems*, SIAM J. Numer. Anal., 30 (1993), pp. 1796–1838.

[70] C. Schittenkopf, G. Deco, and W. Brauer, *Two strategies to avoid overfitting in feedforward networks*, Neural Networks, 10 (1997), pp. 505–516.

[71] J. Schmidhuber, *Discovering neural nets with low Kolmogorov complexicity and high generalization capability*, Neural Networks, 10 (1997), pp. 857–873.

[72] F. Stäger and M. Agarwal, *Three methods to speed up the training of feedforward and feedback perceptrons*, Neural Networks, 10 (1997), pp. 1435–1443.

[73] J. Sum, C. Leung, G. H. Young, L. Chan, and W. Kan, *An adaptive Bayesian pruning for neural networks in a non-stationary environment*, Neural Computation, 11 (1999), pp. 965–976.

[74] S. Tamura and M. Tateishi, *Capabilities of a four-layered feedforward neural network: Four layers versus three*, IEEE Trans. Neural Networks, 8 (1997), pp. 251–255.

[75] I. V. Tetko and A. E. P. Villa, *Efficient partition of learning data sets for neural network training*, Neural Networks, 10 (1997), pp. 1361–1374.

[76] G. Thimm and E. Fiesler, *High-order and multilayer perceptron initialization*, IEEE Trans. Neural Networks, 8 (1997), pp. 349–359.

[77] H. H. Thodberg, *A review of Bayesian neural networks with an application to near infrared spectroscopy*, IEEE Trans. Neural Networks, 7 (1996), pp. 56–72.

[78] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-Posed Problems*, Wiley, New York, 1977.

[79] P. van de Laar and T. Heskes, *Pruning using parameter and neuronal metrics*, Neural Computation, 11 (1999), pp. 977–993.

[80] J. E. Vitela and J. Reifman, *Premature saturation in backpropagation networks: Mechanism and necessary conditions*, Neural Networks, 10 (1997), pp. 721–735.

[81] E. Voudouri-Maniati, L. Kurz, and J. M. Kowalski, *A neural-network approach to nonparametric and robust classification procedures*, IEEE Trans. Neural Networks, 8 (1997), pp. 288–298.

[82] G.-J. Wang and C.-C. Chen, *A fast multilayer neural-network training algorithm based on the layer-by-layer optimizing procedures*, IEEE Trans. Neural Networks, 7 (1996), pp. 768–775.

[83] Y.-J. Wang and C.-T. Lin, *A second-order learning algorithm for multilayer networks based on block Hessian matrix*, Neural Networks, 11 (1998), pp. 1607–1622.

[84] H. White, *Artificial Neural Networks: Approximation and Learning Theory*, Blackwell Publishers, Cambridge, 1992.

[85] S. Yasui, *Convergence suppression and divergence facilitation: Minimum and joint use of hidden units by multiple outputs*, Neural Networks, 10 (1997), pp. 353–367.

[86] J. Zhang and A. J. Morris, *A sequential learning approach for single hidden layer neural networks*, Neural Networks, 11 (1998), pp. 65–80.

[87] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, *L-BFGS-B – Fortran subroutines for large-scale bound constrained optimization*, Report NAM12, Northwestern University EECS, 1995. http://www.ece.nwu.edu/~nocedal/software.html.

Appendix A: Proofs of Lemmas

Lemma 1:
Proof. Functional $J(\mathbf{W})$ in a componentwise form reads as

$$J(\mathbf{W}) = \frac{1}{2} \sum_{i=1}^{m_2} \left(\sum_{j=1}^{m_1} w_{ij}\, v_j - y_i \right)^2 , \tag{28}$$

where i represent the row and j the column index of $\mathbf{W}$, respectively. A straightforward calculation shows that

$$\begin{aligned} \frac{\partial J}{\partial w_{ik}} &= \left(\sum_{j=1}^{m_1} w_{ij}\, v_j - y_i \right) v_k \\ \Longrightarrow \frac{\partial J}{\partial \mathbf{w}_{i,:}} &= \left(\sum_{j=1}^{m_1} w_{ij}\, v_j - y_i \right) \mathbf{v}^T = [\mathbf{W}\,\mathbf{v} - \mathbf{y}]_i\, \mathbf{v}^T \\ \Longrightarrow \frac{\partial J}{\partial \mathbf{W}} &= [\mathbf{W}\,\mathbf{v} - \mathbf{y}]\, \mathbf{v}^T. \end{aligned} \tag{29}$$

Here we have applied a MatLab-type abbreviation $:$ for the row-vector $\mathbf{w}_{i,:}$ consisting of the i'th row of matrix $\mathbf{W}$. This proves the result. □

Lemma 2:
Proof. As before, consider the componentwise form of the functional

$$J(\mathbf{u}) = \frac{1}{2} \sum_{j=1}^{m_2} \left(\sum_{i=1}^{m_1} w_{ji}\, f_i(u_i) - y_j \right)^2 = \frac{1}{2} \sum_{j=1}^{m_2} J_j(\mathbf{u})^2, \tag{30}$$

where we have set $J_j(\mathbf{u}) = \sum_{i=1}^{m_1} w_{ji}\, f_i(u_i) - y_j = [\mathbf{W}\,\mathcal{F}(\mathbf{u}) - \mathbf{y}]_j$. From this we further obtain

$$\frac{\partial J_j}{\partial u_k} = \left[\sum_{i=1}^{m_1} w_{ji}\, f_i(u_i) - y_j \right] w_{jk}\, f_k^{'}(u_k) = w_{jk}\, f_k^{'}(u_k)\, [\mathbf{W}\,\mathcal{F}(\mathbf{u}) - \mathbf{y}]_j. \tag{31}$$

Remember here that $\mathbf{u}$ is a vector with m_2 components. As in Lemma 1, we get the derivative of $J(\mathbf{u})$ with respect to $\mathbf{u}$ by treating k as a row-index and j as a column-index. Proceeding like this, we obtain from (31)

$$\begin{aligned} \nabla_{\mathbf{u}} J(\mathbf{u}) &= \begin{pmatrix} w_{11}\, f_1^{'}(u_1) & w_{21}\, f_1^{'}(u_1) & \dots & w_{m_2 1}\, f_1^{'}(u_1) \\ w_{12}\, f_2^{'}(u_2) & w_{22}\, f_2^{'}(u_2) & \dots & w_{m_2 2}\, f_2^{'}(u_2) \\ \vdots & \vdots & \ddots & \vdots \\ w_{1 m_1}\, f_{m_1}^{'}(u_{m_1}) & w_{2 m_1}\, f_{m_1}^{'}(u_{m_1}) & \dots & w_{m_2 m_1}\, f_{m_1}^{'}(u_{m_1}) \end{pmatrix} [\mathbf{W}\mathcal{F}(\mathbf{u}) - \mathbf{y}] \\ &= \left(\mathbf{W}\,\mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{u})\} \right)^T [\mathbf{W}\,\mathcal{F}(\mathbf{u}) - \mathbf{y}] = \mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{u})\}\, \mathbf{W}^T\, [\mathbf{W}\,\mathcal{F}(\mathbf{u}) - \mathbf{y}], \end{aligned} \tag{32}$$

which is the desired result. □

Corollary 1:
Proof. Introduce yet another index in (30)–(32) to go through the whole function-matrix $\mathcal{F}(u)$. □

Lemma 3:
Proof. Here we introduce the basic Lagrangian technique to simplify the calculations.
As a first step let us introduce an extra variable $\mathbf{u} = \mathbf{W}\mathbf{v}$. Then, instead of problem (17) we consider the equivalent, constraint optimization problem

$$\min_{(\mathbf{u},\mathbf{W})} \tilde{J}(\mathbf{u},\mathbf{W}) = \frac{1}{2}\,\|\bar{\mathbf{W}}\,\mathcal{F}(\mathbf{u}) - \mathbf{y}\|^2 \quad \text{subject to} \quad \mathbf{u} = \mathbf{W}\,\mathbf{v}, \tag{33}$$

where $\mathbf{u}$ and $\mathbf{W}$ are treated as *independent* variables which are linked together by the given constraint. The Lagrange-functional associated with (33) reads as

$$\mathcal{L}(\mathbf{u},\mathbf{W},\boldsymbol{\lambda}) = \tilde{J}(\mathbf{u},\mathbf{W}) + \boldsymbol{\lambda}^T(\mathbf{u} - \mathbf{W}\,\mathbf{v}), \tag{34}$$

where $\boldsymbol{\lambda} \in\in \mathbb{R}^{m_1}$ contains the Lagrangian variables for the constraint.
Noticing that the values of functions $\tilde{J}(\mathbf{u},\mathbf{W})$ in (33) and $\mathcal{L}(\mathbf{u},\mathbf{W},\boldsymbol{\lambda})$ in (34) coincide if the constraint $\mathbf{u} = \mathbf{W}\,\mathbf{v}$ is satisfied, it follows that solution of (33) is equivalently characterized by the saddle-point conditions $\nabla\mathcal{L}(\mathbf{u},\mathbf{W}^1,\boldsymbol{\lambda}) = 0$. Our technique here is based on using these saddle-points conditions to remove the extra unknowns $\mathbf{u}$ and $\boldsymbol{\lambda}$ (if and when they are well-defined, which in the optimization theory refers to the concept of constraint qualification) for obtaining the desired gradient $\nabla_{\mathbf{W}}J(\mathbf{W})$. Therefore, we compute the derivatives $\nabla_{\mathbf{u}}\mathcal{L}$, $\nabla_{\mathbf{W}}\mathcal{L}$ and $\nabla_{\boldsymbol{\lambda}}\mathcal{L}$ (in vector- and matrix-form) for the Lagrangian in (34) next.
The gradient-vector $\nabla_{\mathbf{u}}\mathcal{L}(\mathbf{u},\mathbf{W},\boldsymbol{\lambda})$ is, due to Lemma 2, of the form

$$\nabla_{\mathbf{u}}\mathcal{L} = \nabla_{\mathbf{u}}\tilde{J}(\mathbf{u}) + \boldsymbol{\lambda} = \mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{u})\}\,\bar{\mathbf{W}}^T\,[\mathbf{W}\,\mathcal{F}(\mathbf{u}) - \mathbf{y}] + \boldsymbol{\lambda}. \tag{35}$$

Derivates with respect to other unknowns are given by

$$\begin{aligned} \nabla_{\mathbf{W}}\mathcal{L} &= -\boldsymbol{\lambda}\,\mathbf{v}^T, & (36)\\ \nabla_{\boldsymbol{\lambda}}\mathcal{L} &= \mathbf{u} - \mathbf{W}\mathbf{v}. & (37) \end{aligned}$$

Using formula $-\boldsymbol{\lambda} = \mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{u})\}\,\bar{\mathbf{W}}^T\,[\mathbf{W}\,\mathcal{F}(\mathbf{u}) - \mathbf{y}]$ due to (35) and substituting this into (36) we obtain

$$\nabla_{\mathbf{W}}\mathcal{L} = \mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{u})\}\,\bar{\mathbf{W}}^T\,[\bar{\mathbf{W}}\,\mathcal{F}(\mathbf{u}) - \mathbf{y}]\,\mathbf{v}^T. \tag{38}$$

Finally, when $\mathbf{u}$ is replaced with $\mathbf{W}\mathbf{v}$ obtained from (37), (38) can be written as

$$\mathrm{Diag}\{\mathcal{F}^{'}(\mathbf{W}\,\mathbf{v})\}\,\bar{\mathbf{W}}^T\,[\bar{\mathbf{W}}\,\mathcal{F}(\mathbf{W}\,\mathbf{v}) - \mathbf{y}]\,\mathbf{v}^T, \tag{39}$$

which is represented only using the unknown matrix $\mathbf{W}$. Due to the equivalency of (17), (33) and (34), (39) gives the desired gradient-matrix for (17). This ends the proof. $\square$

Lemma 4:
Proof. To simplify the calculations, we now introduce two extra variables $\mathbf{u} = \mathbf{W}\mathbf{v}$ and $\tilde{\mathbf{u}} = \tilde{\mathbf{W}}\,\tilde{\mathcal{F}}(\mathbf{u}) = \tilde{\mathbf{W}}\,\tilde{\mathcal{F}}(\mathbf{W}\mathbf{v})$. As in Lemma 3, instead of problem (20) we consider the equivalent, constraint optimization problem

$$\min_{(\mathbf{u},\tilde{\mathbf{u}},\mathbf{W})} \tilde{J}(\mathbf{u},\tilde{\mathbf{u}},\mathbf{W}) = \frac{1}{2}\,\|\bar{\mathbf{W}}\,\bar{\mathcal{F}}(\tilde{\mathbf{u}}) - \mathbf{y}\|^2 \quad \text{subject to} \quad \mathbf{u} = \mathbf{W}\,\mathbf{v} \text{ and } \tilde{\mathbf{u}} = \tilde{\mathbf{W}}\,\tilde{\mathcal{F}}(\mathbf{u}). \tag{40}$$

The Lagrange-functional associated with (40) reads as

$$\mathcal{L}(\mathbf{u},\tilde{\mathbf{u}},\mathbf{W},\boldsymbol{\lambda},\tilde{\boldsymbol{\lambda}}) = \tilde{J}(\mathbf{u},\tilde{\mathbf{u}},\mathbf{W}) + \boldsymbol{\lambda}^T(\mathbf{u} - \mathbf{W}\,\mathbf{v}) + \tilde{\boldsymbol{\lambda}}^T(\tilde{\mathbf{u}} - \tilde{\mathbf{W}}\,\tilde{\mathcal{F}}(\mathbf{u})). \tag{41}$$

Using similar techniques as in the previous proof, the derivates realizing the saddle-point conditions for the Lagrangian are given by

$$\begin{aligned}
\nabla_{\mathbf{u}}\mathcal{L} &= -[\tilde{\mathbf{W}}\,\tilde{\mathcal{F}}'(\mathbf{u})]^T\,\tilde{\boldsymbol{\lambda}} + \boldsymbol{\lambda}, && (42)\\
\nabla_{\tilde{\mathbf{u}}}\mathcal{L} &= [\bar{\mathbf{W}}\,\bar{\mathcal{F}}'(\tilde{\mathbf{u}})]^T\,[\bar{\mathbf{W}}\,\bar{\mathcal{F}}(\tilde{\mathbf{u}}) - \mathbf{y}] + \tilde{\boldsymbol{\lambda}}, && (43)\\
\nabla_{\mathbf{W}}\mathcal{L} &= -\boldsymbol{\lambda}\,\mathbf{v}^T, && (44)\\
\nabla_{\boldsymbol{\lambda}}\mathcal{L} &= \mathbf{u} - \mathbf{W}\mathbf{v}, && (45)\\
\nabla_{\tilde{\boldsymbol{\lambda}}}\mathcal{L} &= \tilde{\mathbf{u}} - \tilde{\mathbf{W}}\,\tilde{\mathcal{F}}(\mathbf{u}). && (46)
\end{aligned}$$

From $\nabla_{(\mathbf{u},\tilde{\mathbf{u}})}\mathcal{L} = \mathbf{O}$ and (42)–(43) we obtain

$$\boldsymbol{\lambda} = [\tilde{\mathbf{W}}\,\tilde{\mathcal{F}}'(\mathbf{u})]^T\,\tilde{\boldsymbol{\lambda}} = -[\tilde{\mathbf{W}}\,\tilde{\mathcal{F}}'(\mathbf{u})]^T\,[\bar{\mathbf{W}}\,\bar{\mathcal{F}}'(\tilde{\mathbf{u}})]^T\,[\bar{\mathbf{W}}\,\bar{\mathcal{F}}(\tilde{\mathbf{u}}) - \mathbf{y}]. \tag{47}$$

Substituting this into $\nabla_{\mathbf{W}}\mathcal{L}$ in (44) yields

$$\nabla_{\mathbf{W}}\mathcal{L} = \mathrm{Diag}\{\tilde{\mathcal{F}}'(\mathbf{u})\}\,\tilde{\mathbf{W}}^T\,\mathrm{Diag}\{\bar{\mathcal{F}}'(\tilde{\mathbf{u}})\}\,\bar{\mathbf{W}}^T\,[\bar{\mathbf{W}}\,\bar{\mathcal{F}}(\tilde{\mathbf{u}}) - \mathbf{y}]\,\mathbf{v}^T. \tag{48}$$

This together with the original expressions $\mathbf{u} = \mathbf{W}\,\mathbf{v}$ and $\tilde{\mathbf{u}} = \tilde{\mathbf{W}}\,\tilde{\mathcal{F}}(\mathbf{u})$ proves the result. $\square$

Appendix B: Figures

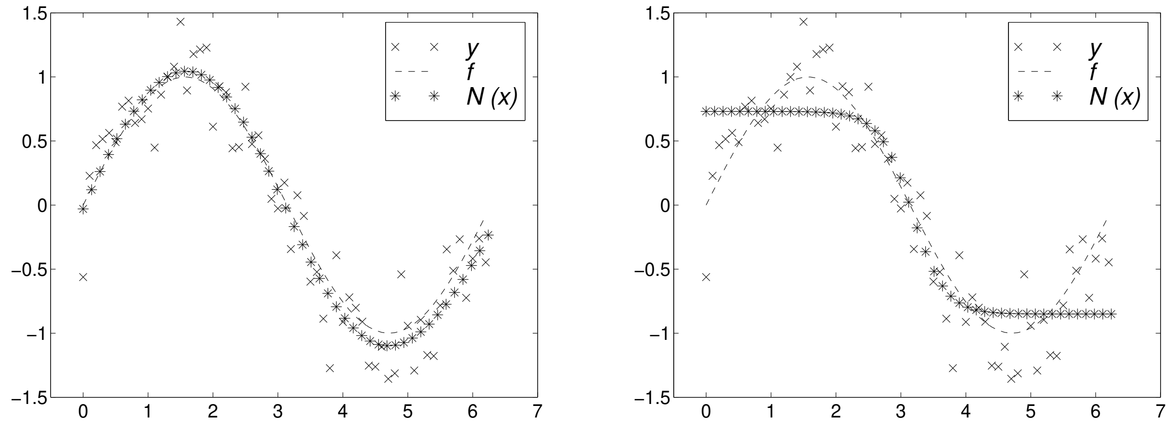

Figure 5: $n_1 = 2$ in Table 1. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

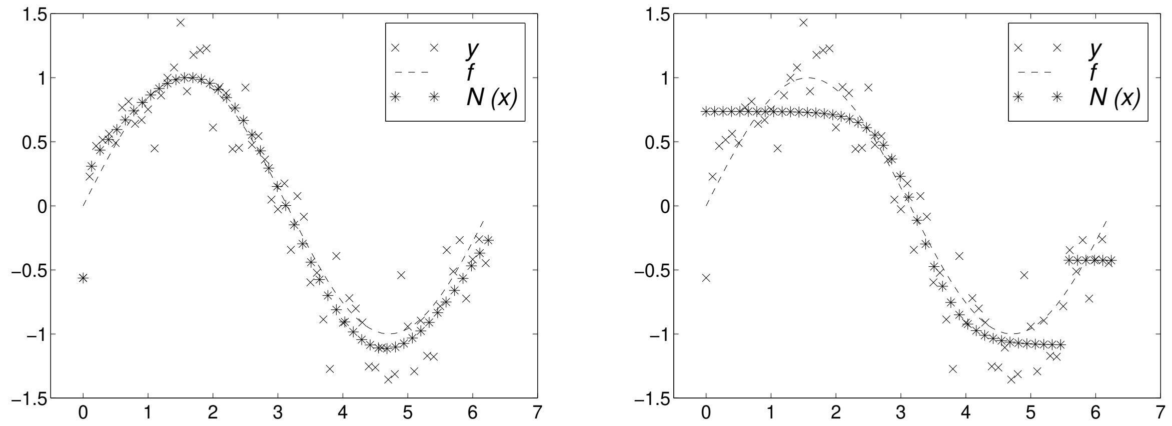

Figure 6: $n_1 = 3$ in Table 1. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

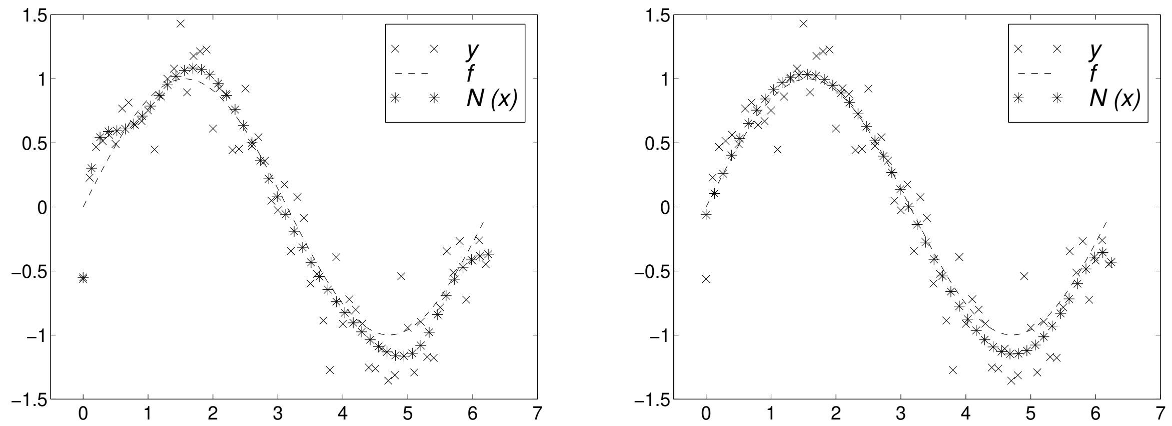

Figure 7: $n_1 = 4$ in Table 1. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

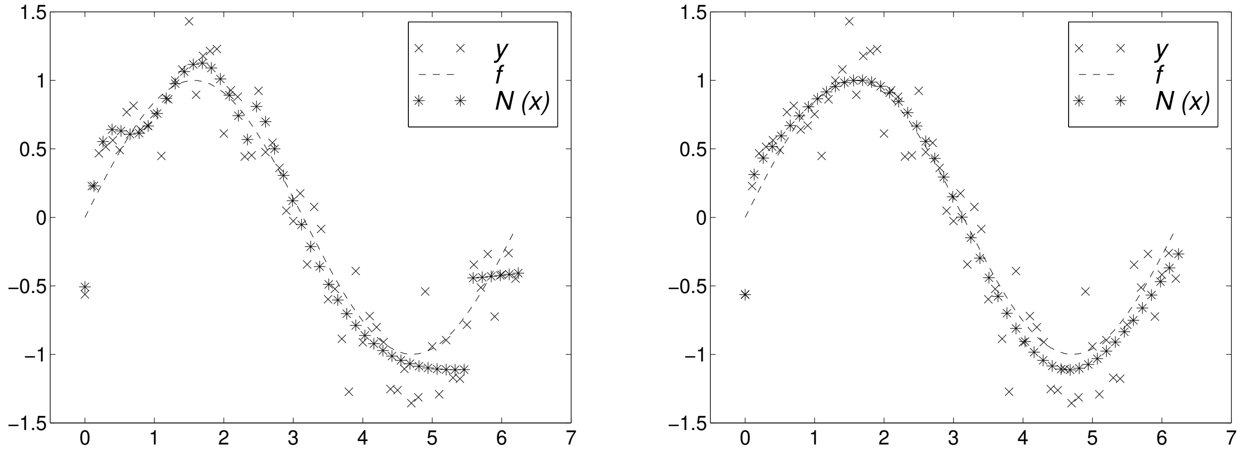

Figure 8: $n_1 = 5$ in Table 1. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

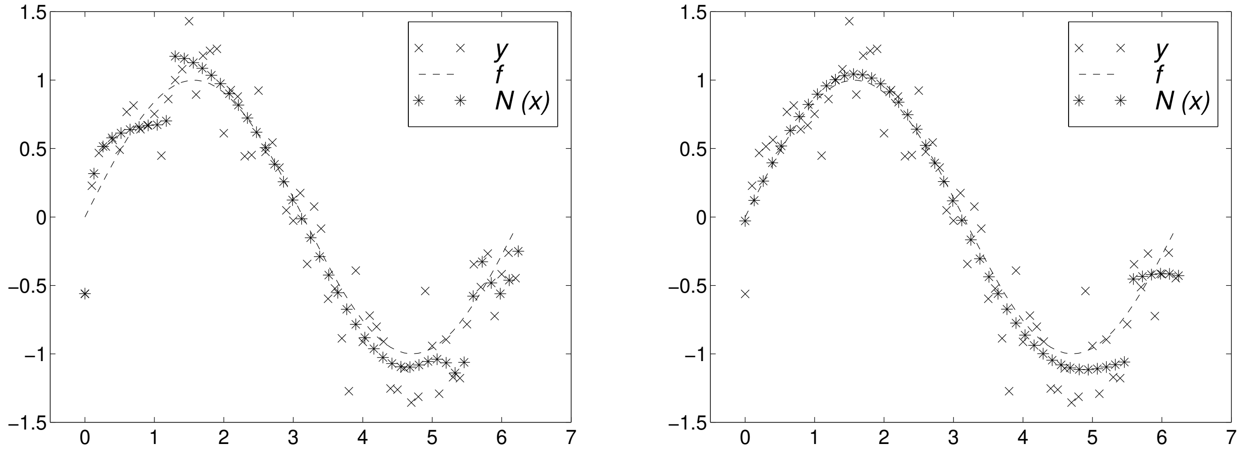

Figure 9: $n_1 = 6$ in Table 1. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

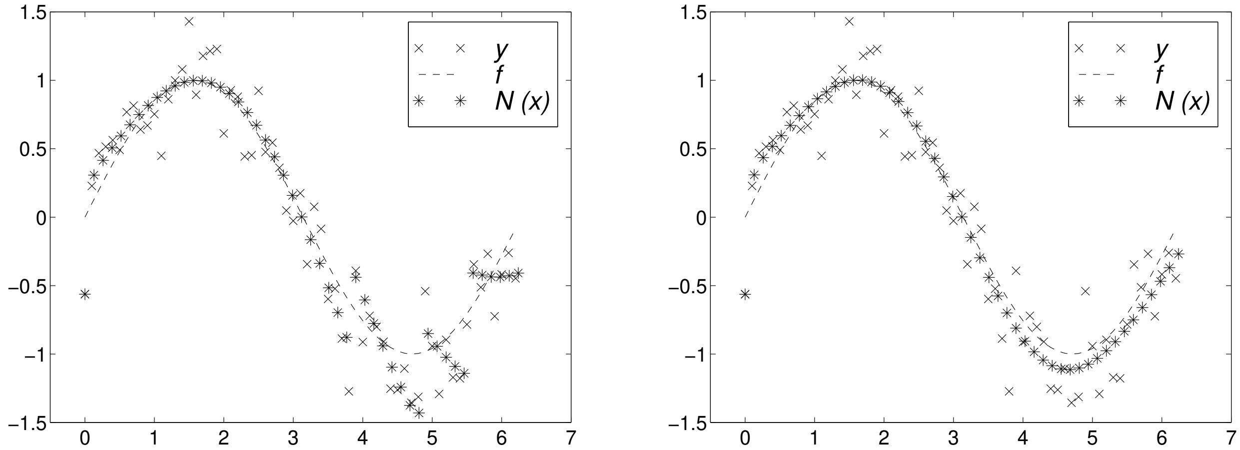

Figure 10: $n_1 = 7$ in Table 1. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

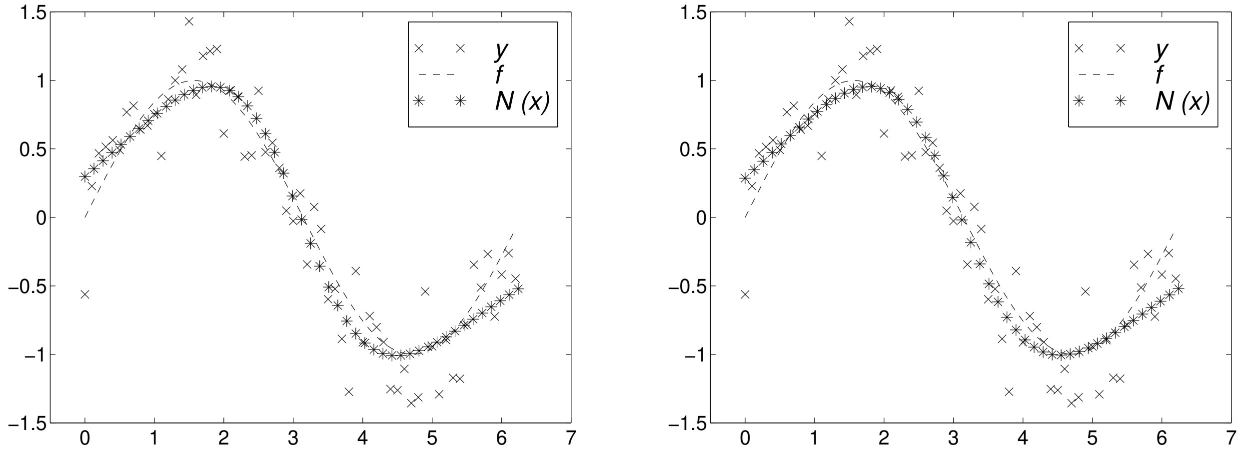

Figure 11: $n_1 = 2$ in Table 2. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

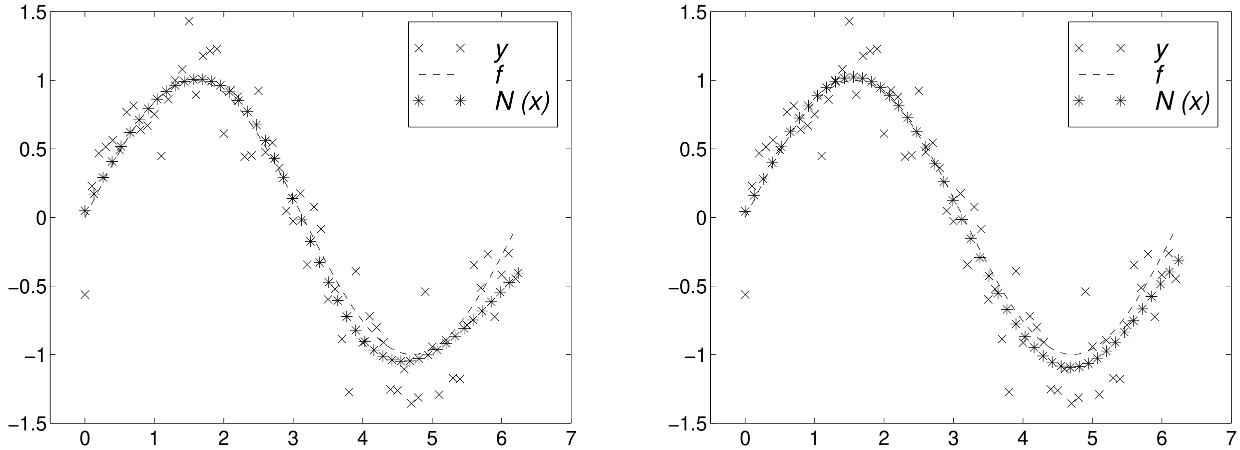

Figure 12: $n_1 = 3$ in Table 2. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

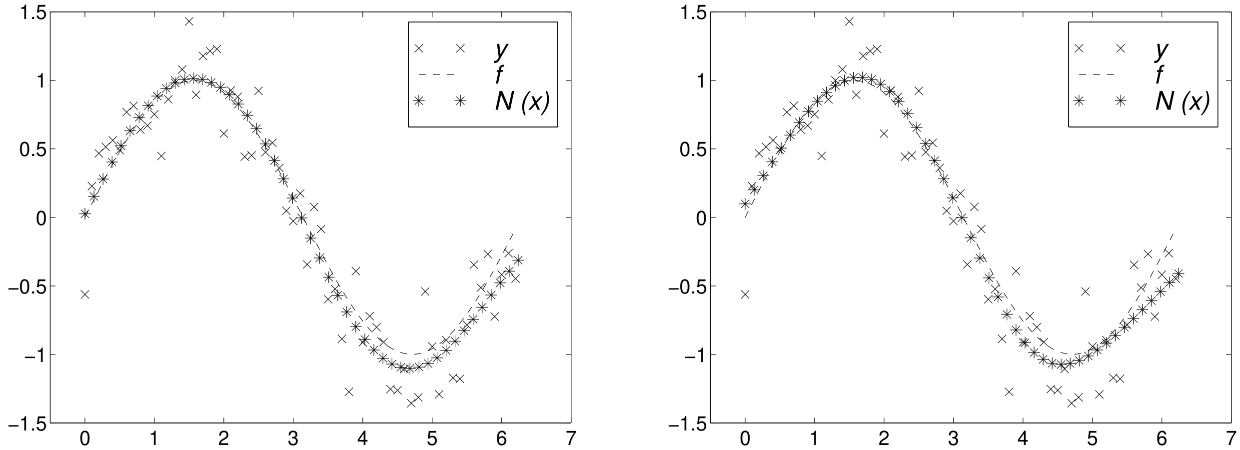

Figure 13: $n_1 = 4$ in Table 2. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

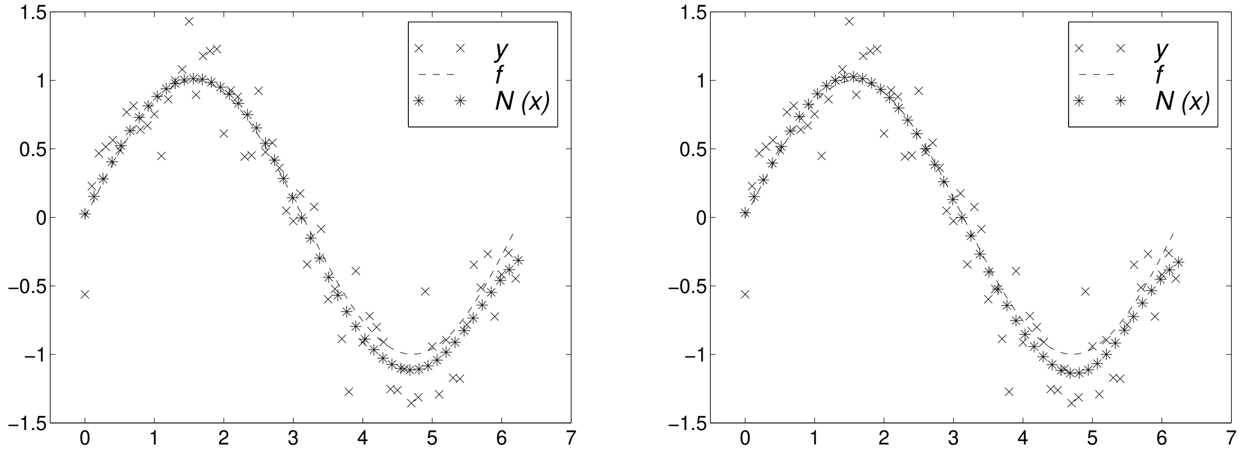

Figure 14: $n_1 = 5$ in Table 2. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

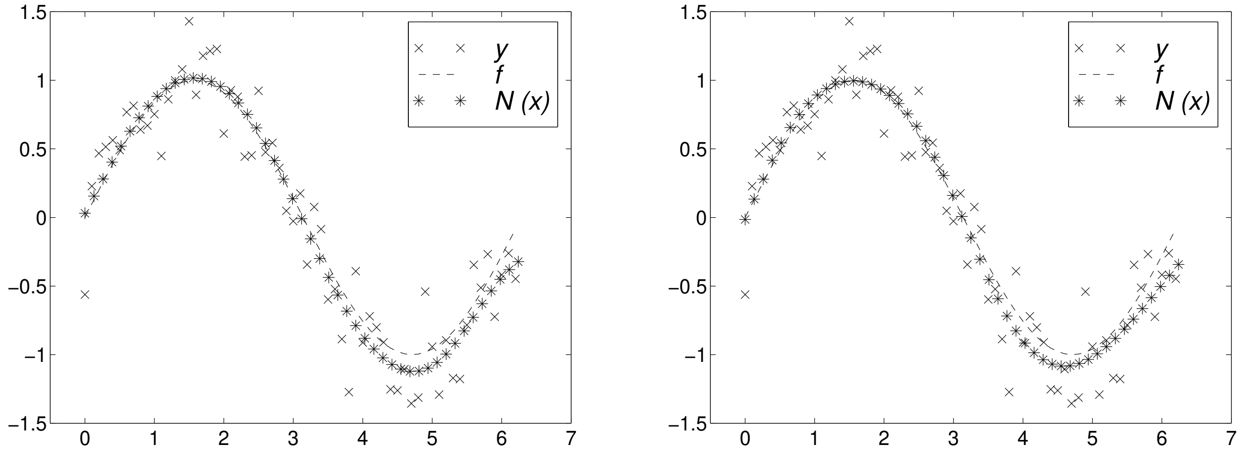

Figure 15: $n_1 = 6$ in Table 2. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

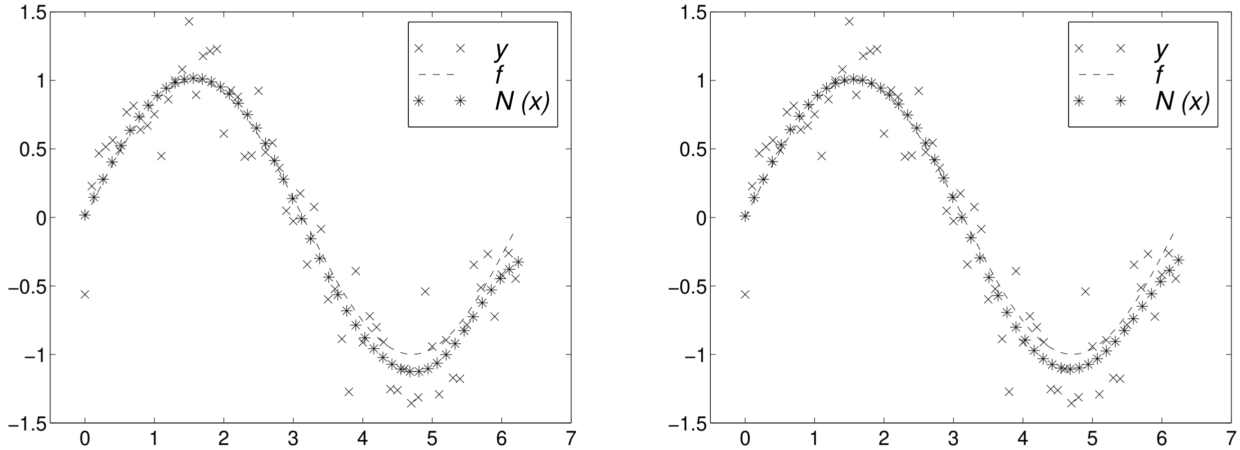

Figure 16: $n_1 = 7$ in Table 2. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

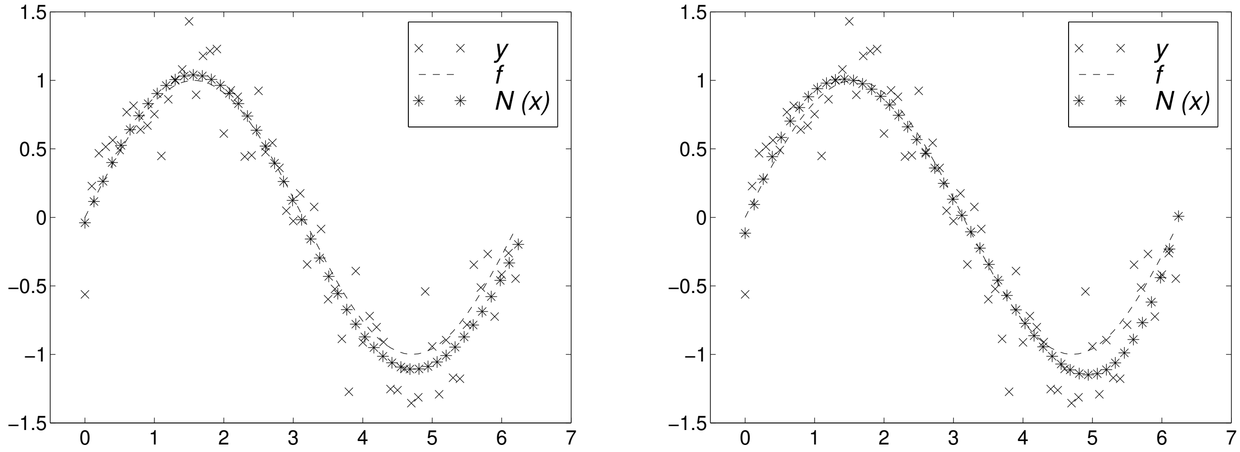

Figure 17: $n_1 = 2$ in Table 3. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

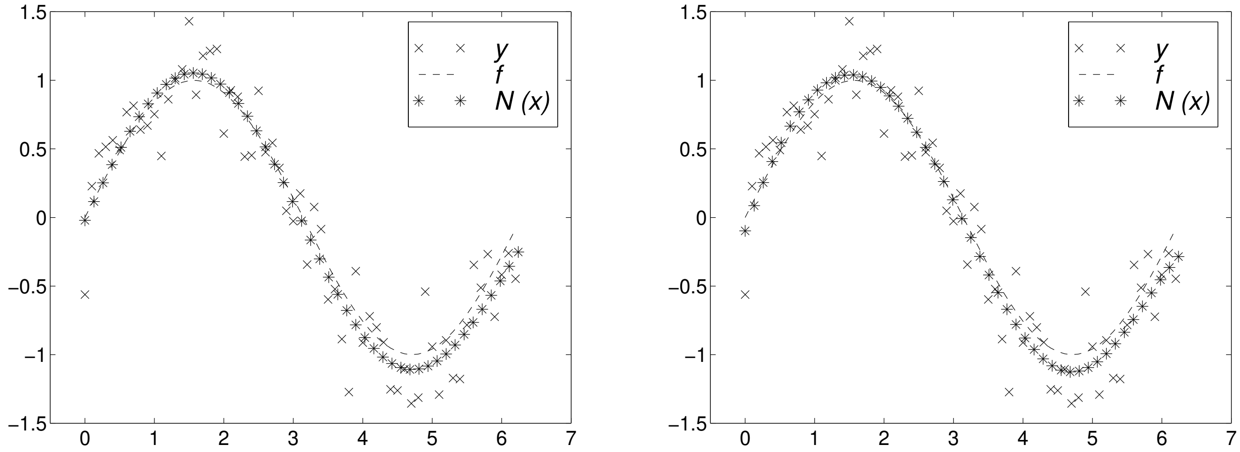

Figure 18: $n_1 = 3$ in Table 3. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

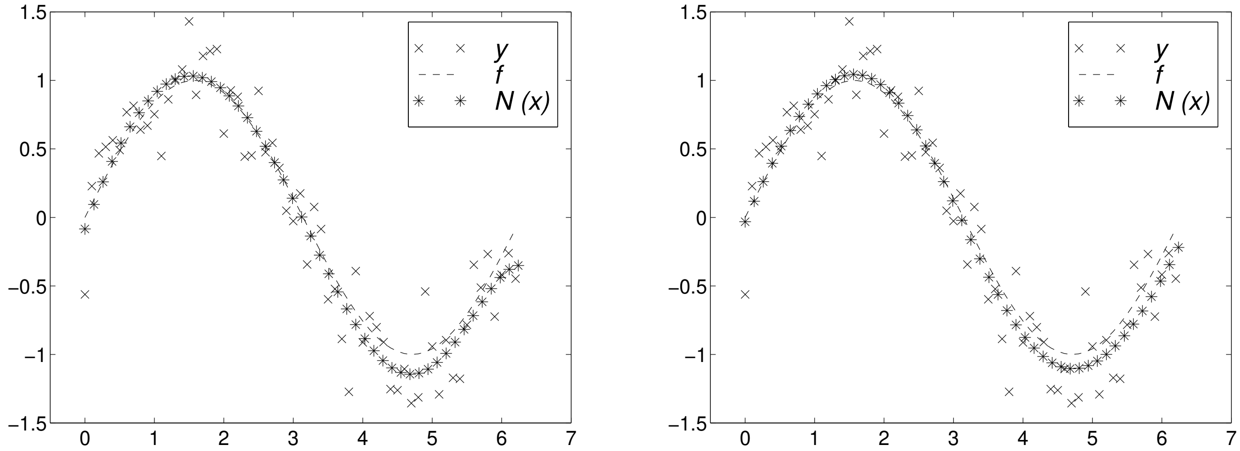

Figure 19: $n_1 = 4$ in Table 3. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

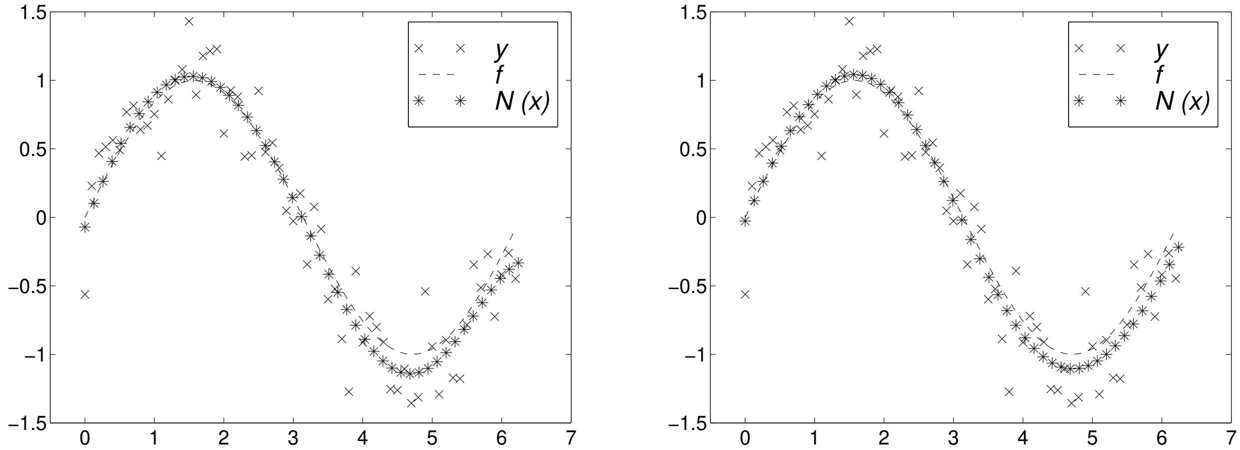

Figure 20: $n_1 = 5$ in Table 3. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

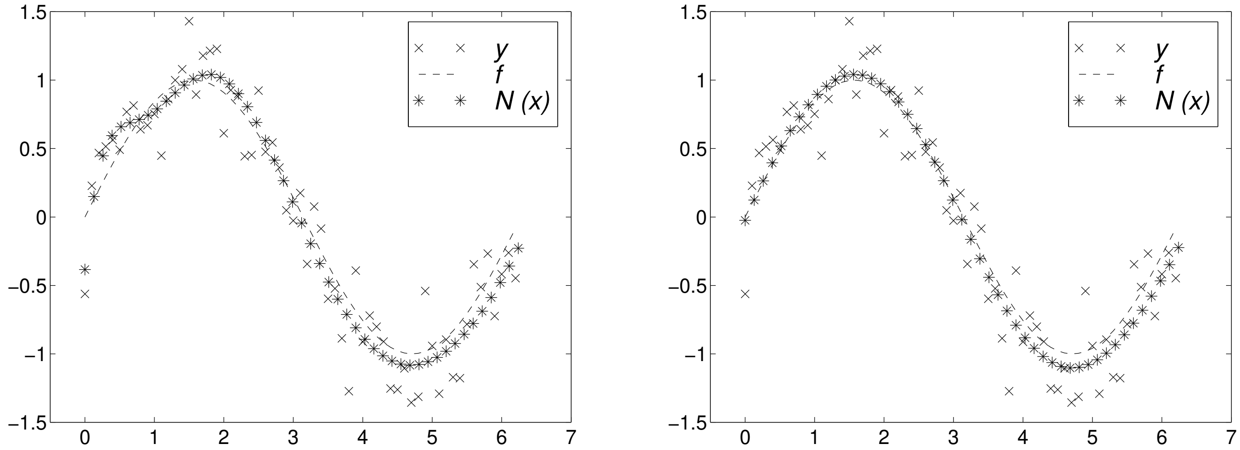

Figure 21: $n_1 = 6$ in Table 3. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

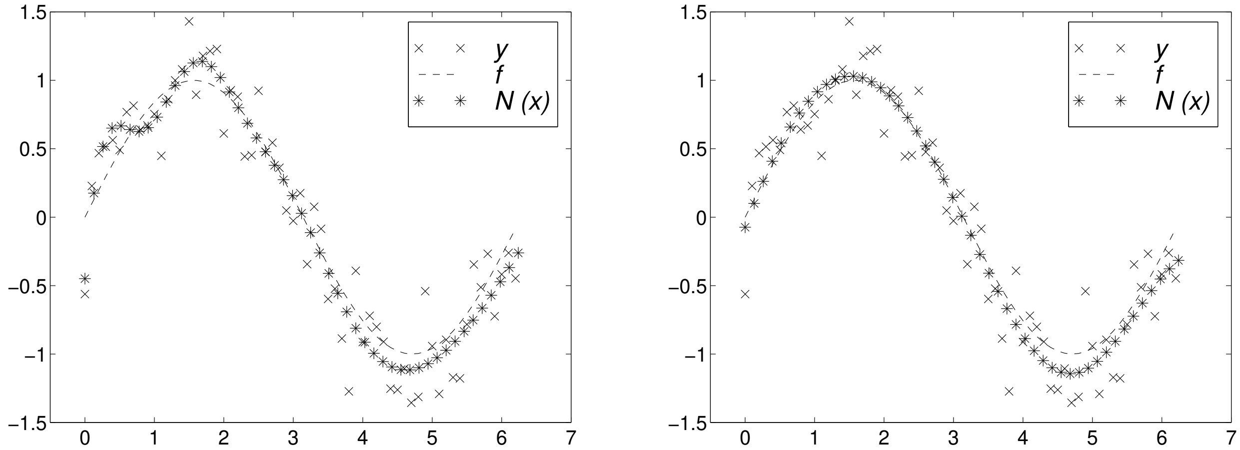

Figure 22: $n_1 = 7$ in Table 3. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

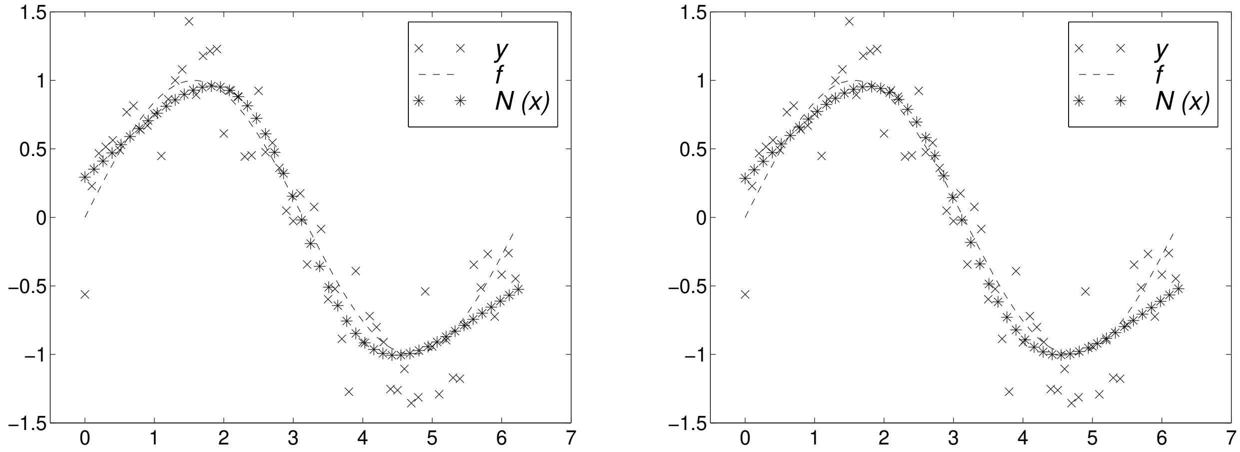

Figure 23: $n_1 = 2$ in Table 4. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

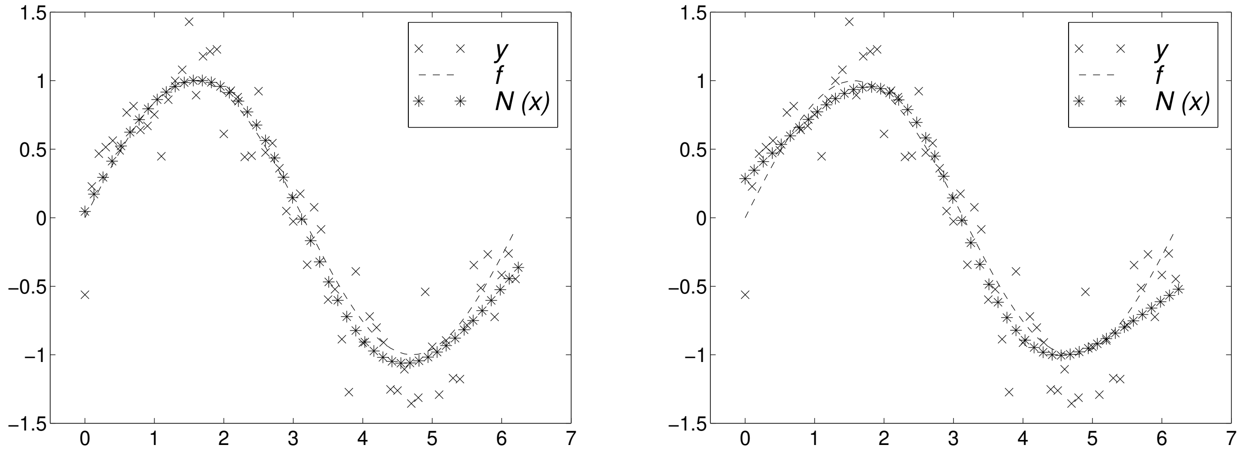

Figure 24: $n_1 = 3$ in Table 4. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

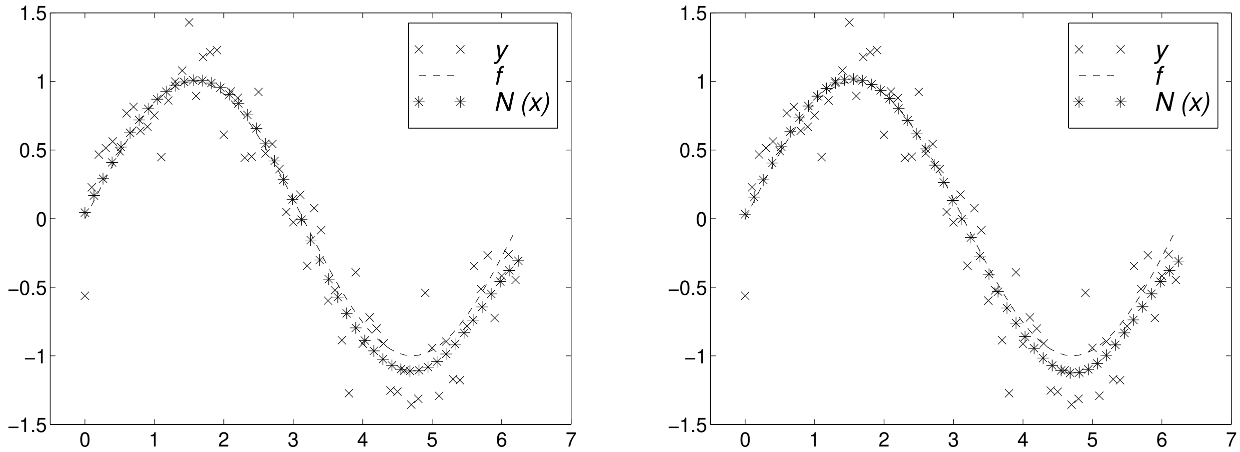

Figure 25: $n_1 = 4$ in Table 4. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

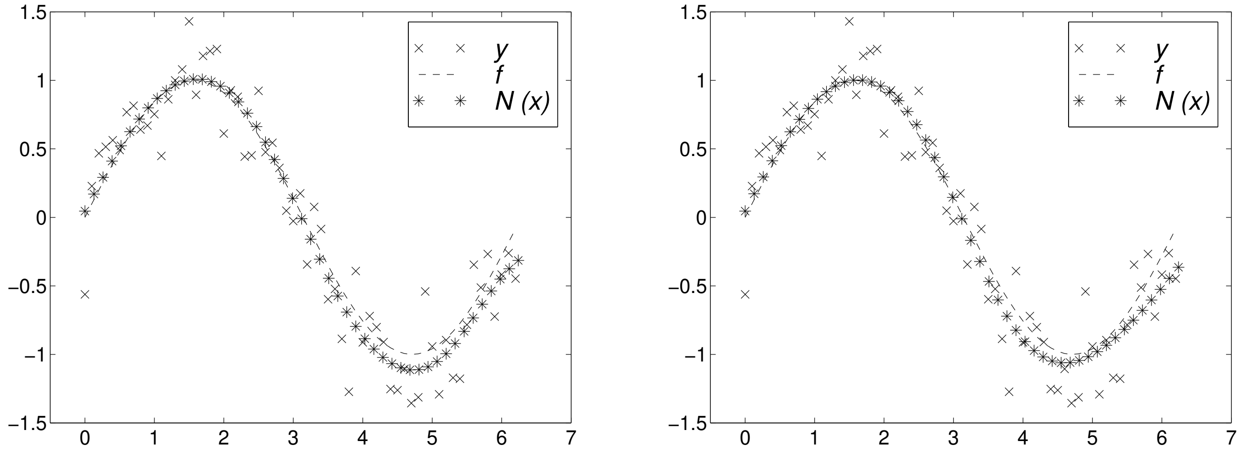

Figure 26: $n_1 = 5$ in Table 4. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

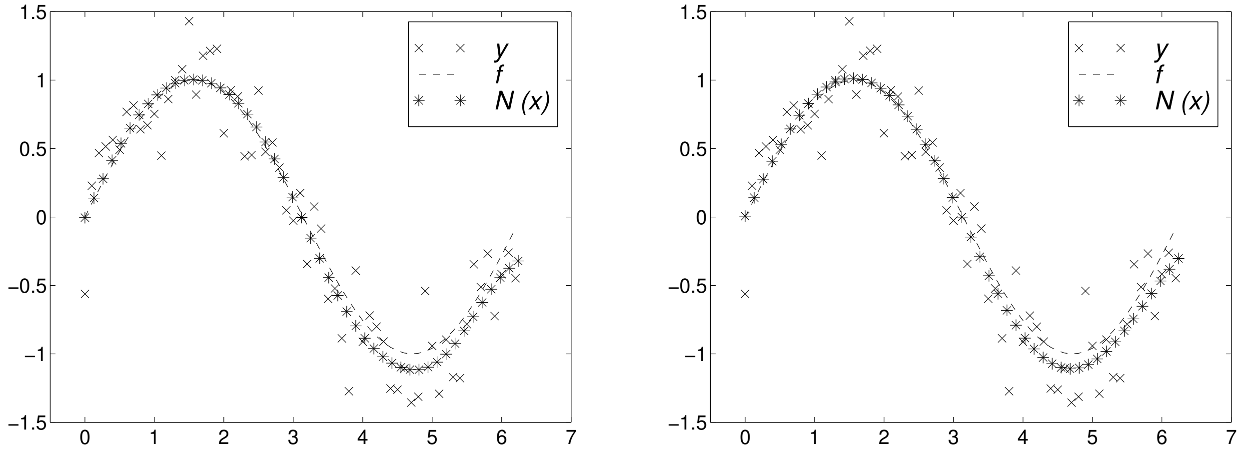

Figure 27: $n_1 = 6$ in Table 4. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

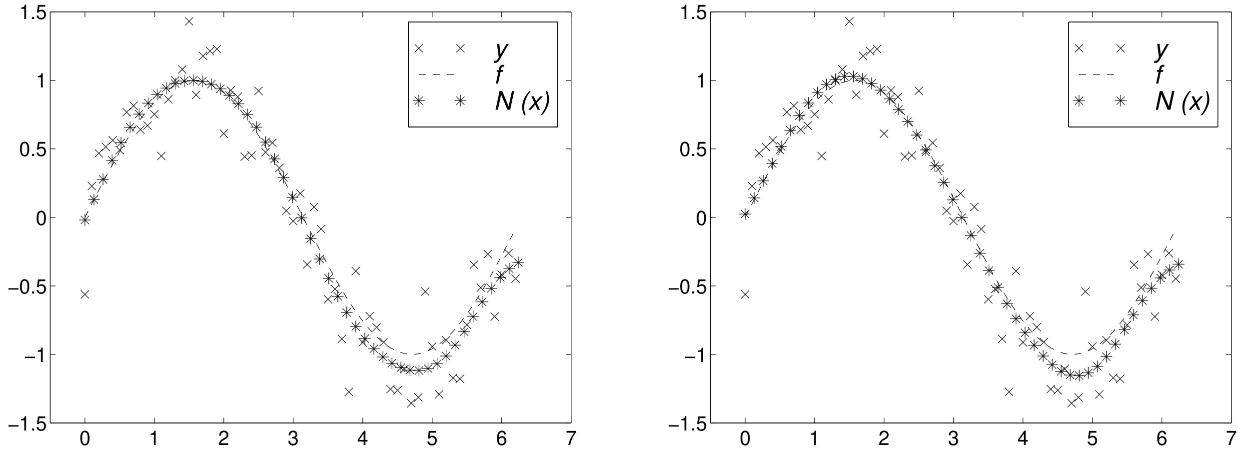

Figure 28: $n_1 = 7$ in Table 4. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

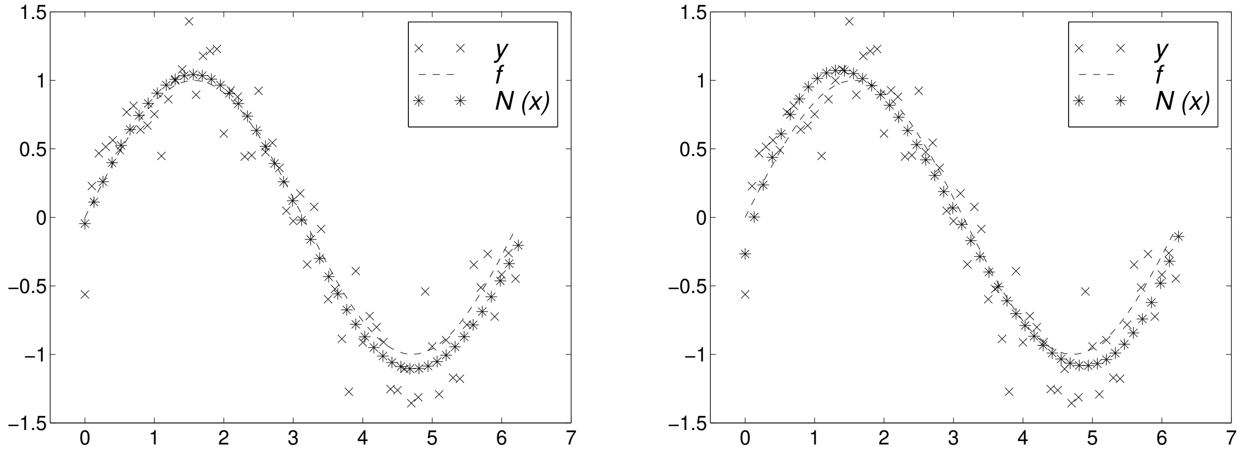

Figure 29: $n_1 = 2$ in Table 5. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

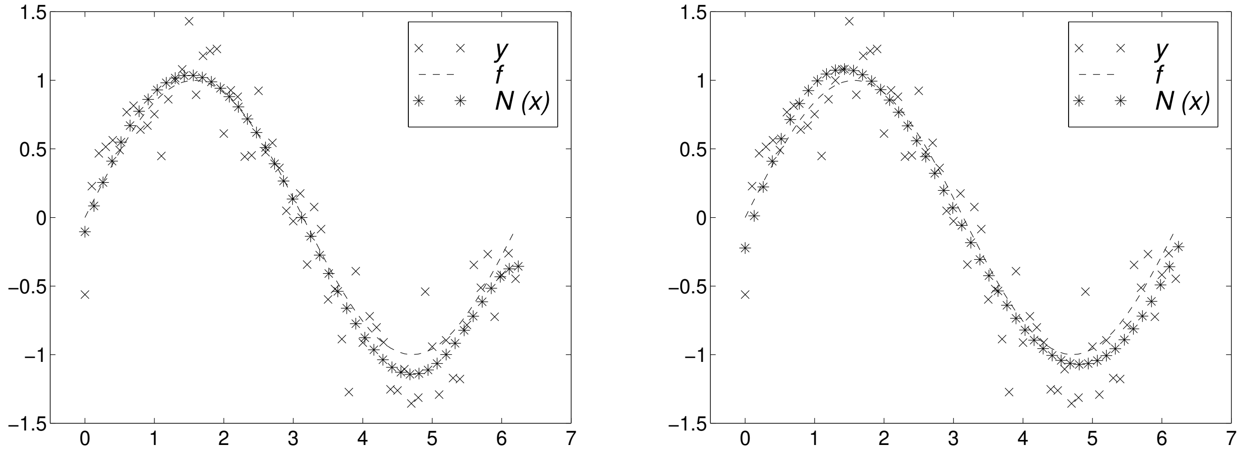

Figure 30: $n_1 = 3$ in Table 5. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

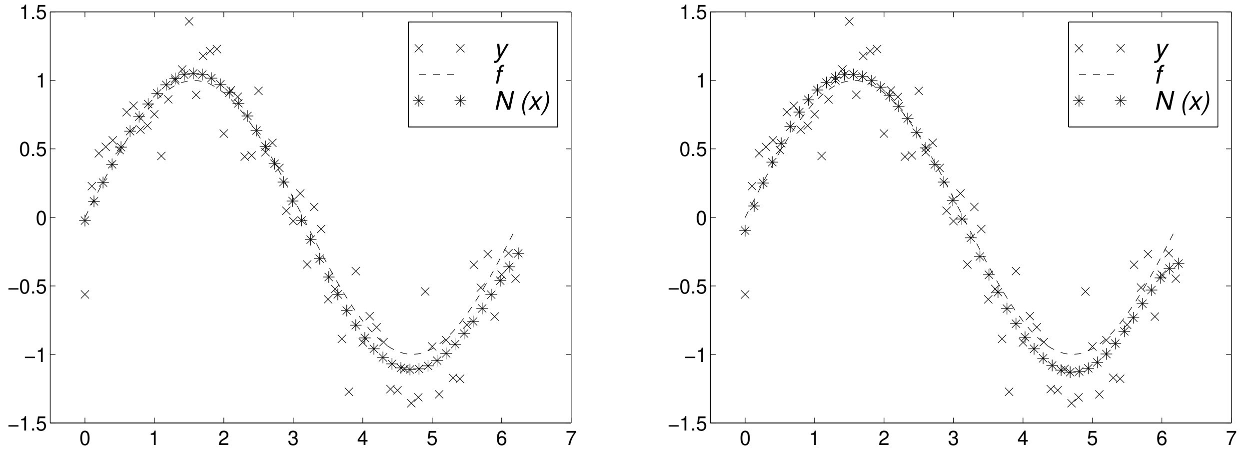

Figure 31: $n_1 = 4$ in Table 5. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

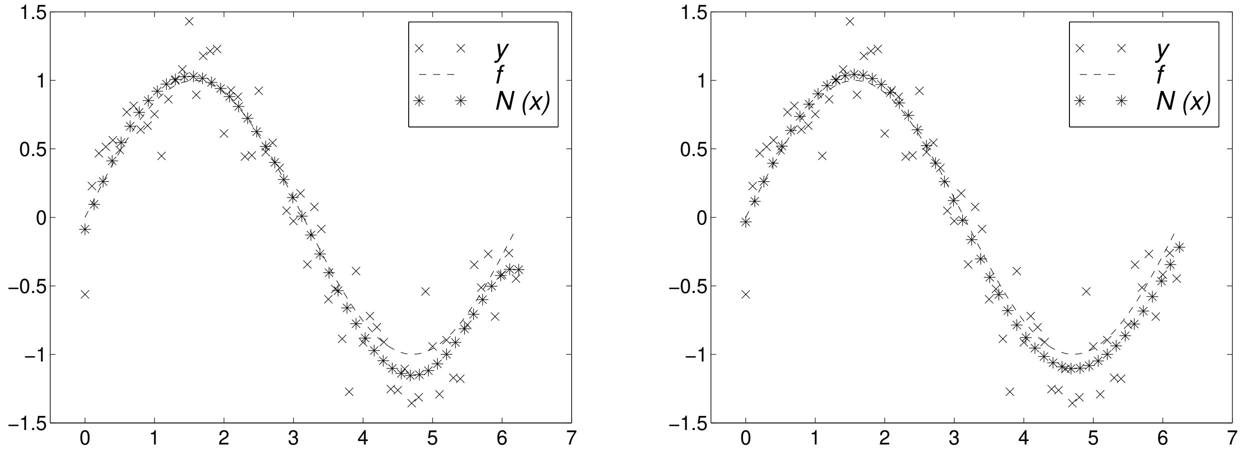

Figure 32: $n_1 = 5$ in Table 5. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

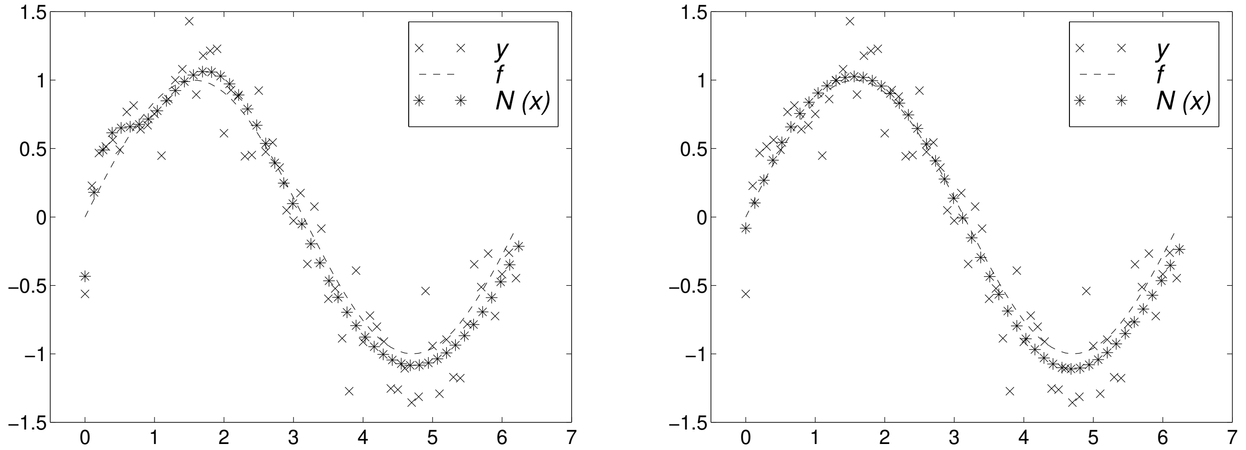

Figure 33: $n_1 = 6$ in Table 5. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

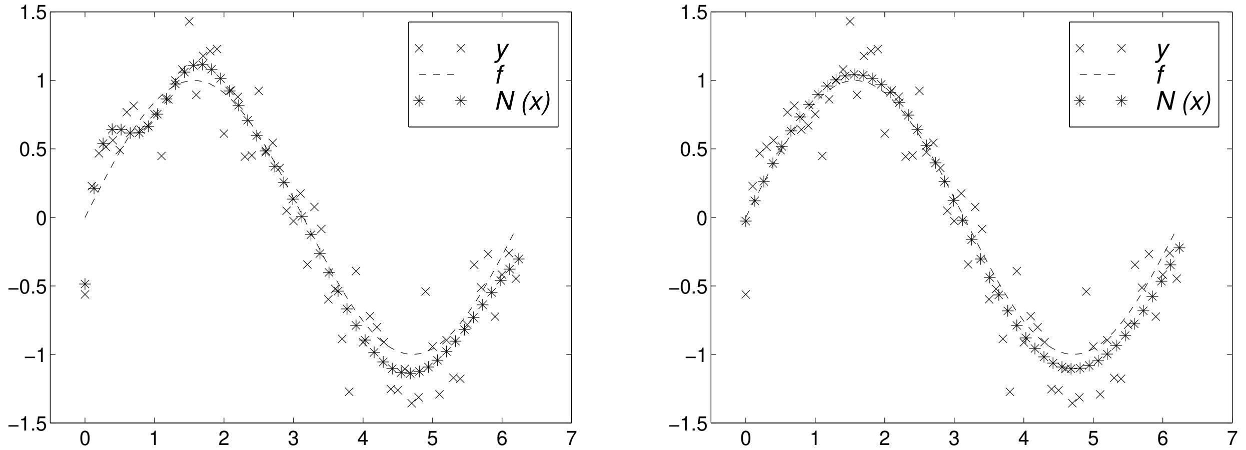

Figure 34: $n_1 = 7$ in Table 5. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

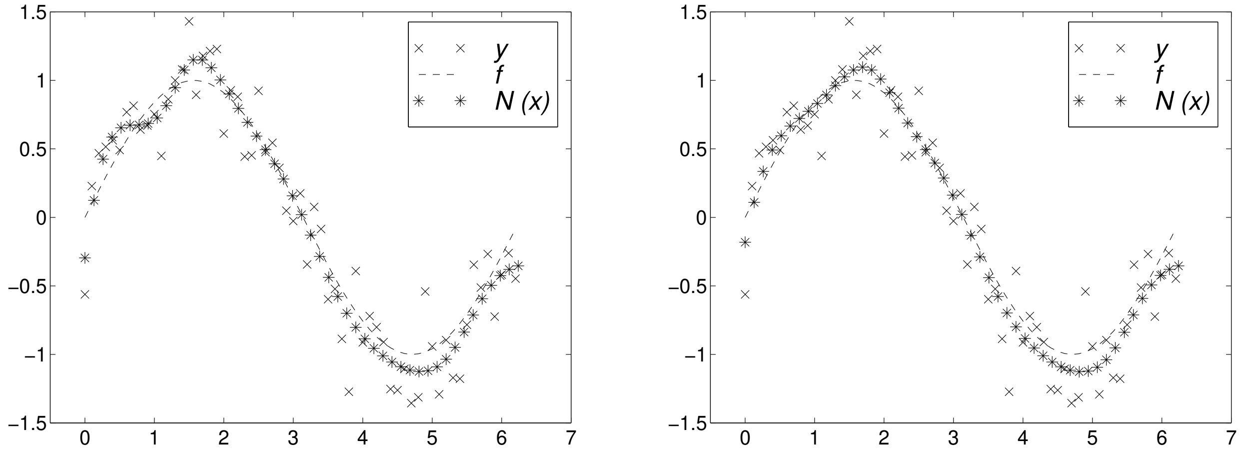

Figure 35: $n_1 = 15$ for regularization method **I**. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).

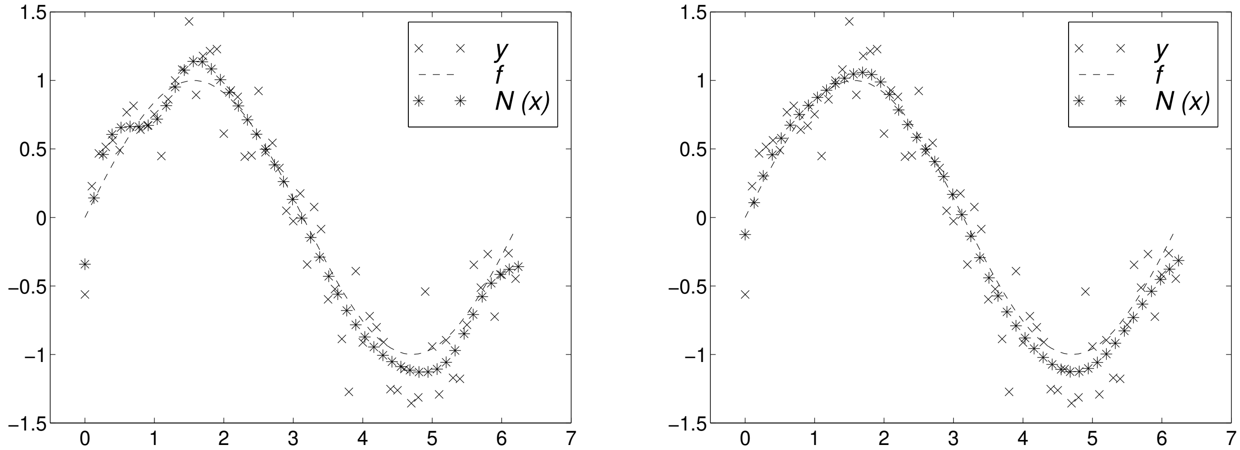

Figure 36: $n_1 = 15$ for regularization method **III**. Results for minimum of $\mathcal{J}^*$ (left) and maximum of $\mathcal{J}^*$ (right).